Statistical Semantic Processing using Markov Logic

Ivan Vladimir Meza-Ruiz

University of Edinburgh

Doctor of Philosophy

Institute for Communicating and Collaborative Systems

School of Informatics

University of Edinburgh

2009

Abstract

Markov Logic (ML) is a novel approach to Natural Language Processing tasks [Richardson and Domingos, 2006; Riedel, 2008]. It is a Statistical Relational Learning language based on First Order Logic (FOL) and Markov Networks (MN). It allows one to treat a task as structured classification. In this work, we investigate ML for the semantic processing tasks of Spoken Language Understanding (SLU) and Semantic Role Labelling (SRL). Both tasks consist of identifying a semantic representation for the meaning of a given utterance/sentence. However, they differ in nature: SLU is in the field of dialogue systems where the domain is closed and language is spoken [He and Young, 2005], while SRL is for open domains and traditionally for written text [Márquez et al., 2008].

Robust SLU is a key component of spoken dialogue systems. This component consists of identifying the meaning of the user utterances addressed to the system. Recent statistical approaches to SLU depend on additional resources (e.g., gazetteers, grammars, syntactic treebanks) which are expensive and time-consuming to produce and maintain. On the other hand, simple datasets annotated only with slot-values are commonly used in dialogue system development, and are easy to collect, automatically annotate, and update. However, slot-values leave out some of the fine-grained long distance dependencies present in other semantic representations. In this work we investigate the development of SLU modules with minimum resources with slot-values as their semantic representation. We propose to use the ML to capture long distance dependencies which are not explicitly available in the slot-value semantic representation. We test the adequacy of the ML framework by comparing against a set of baselines using state of the art approaches to semantic processing. The results of this research have been published in Meza-Ruiz et al. [2008a,b].

Furthermore, we address the question of scalability of the ML approach for other NLP tasks involving the identification of semantic representations. In particular, we focus on SRL: the task of identifying predicates and arguments within sentences, together with their semantic roles. The semantic representation built during SRL is more complex than the slot-values used in dialogue systems, in the sense that they include the notion of predicate/argument scope. SRL is defined in the context of open domains under the premises that there are several levels of extra resources (lemmas, POS tags, constituent or dependency parses). In this work, we propose a ML model of SRL and experiment with the different architectures we can describe for the model which gives

us an insight into the types of correlations that the ML model can express [Riedel and Meza-Ruiz, 2008; Meza-Ruiz and Riedel, 2009].

Additionally, we tested our minimal resources setup in a state of the art dialogue system: the TownInfo system. In this case, we were given a small dataset of gold standard semantic representations which were system dependent, and we rapidly developed a SLU module used in the functioning dialogue system. No extra resources were necessary in order to reach state of the art results.

Acknowledgements

First I want to thank my supervisors Oliver Lemon, Alex Lascarides and Jamie Henderson for their support, feedback and patience. I am eternally grateful to Sebastian Riedel for his help, supervision, support, time and friendship. I was happy during my studies because I had a home in Edinburgh, for that I thank Katharine and Steven McCullagh, and Hannah Beaven, and their families which show me a great deal about the kindness of the Scottish and Irish, in particular Sheila Beaven. I want to thank the friendship and support from my collogues: Pei-Yun (Sabrina) Hsueh, Andi Winterboer, Heriberto Cuayahuitl, Le Zhang, Hieu Hoang, Abhishek Arun and Loic Dugast. To the Mexicans in Edinburgh, the football group and the Bannerman's bunch, thank you for the good times. Finally, this would not have been possible without the support of my family in my Mexican ground, the wise advice of Luis Pineda and the economic support from my sponsor, CONACYT.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ivan Vladimir Meza-Ruiz*
University of Edinburgh)

To mom, dad, Rolan, Tania, Tio Pepe, Katharine and Steven McCullagh

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Markov Logic [ML, Richardson and Domingos, 2006] is a Statistical Relational Learning language based on First Order Logic (FOL) and Markov Networks (MN). In a similar way to other machine learning frameworks (e.g., MaxEnt, Conditional Random Fields), ML separates the modelling of a task from the machine learning process. FOL is used to describe the relations between elements of the task. These relations are transformed into MNs which are used to create statistical models of the task. In this thesis, we focus on the design of ML models for two semantic processing tasks: Spoken Language Understanding (SLU) [He and Young, 2005] and Semantic Role Labelling (SRL) [Márquez et al., 2008]. Both tasks consist of identifying a semantic representation of the meaning of an utterance/sentence.

The outline of the rest of this chapter is as follows. Section 1.1 presents the SLU and SRL task. Section 1.2 presents the different aspects of the semantic representations used for dialogue systems and SRL. Section 1.3 introduces the measurement metrics we use to evaluate the experiments we perform in this work. Section 1.4 presents the contributions of this work. Finally, section 1.5 presents the outline of the rest of the thesis.

1.1 Semantic processing and Markov Logic

A semantic processing task maps an utterance/sentence into a semantic representation. Spoken Language Understanding (SLU) and Semantic Role Labelling (SRL) are two types of semantic processing task. SLU is part of a dialogue system understanding stage, therefore uses spoken language (i.e., utterances). In contrast, SRL uses written language (i.e. sentences). Recent development of both tasks focuses on data oriented

approaches. In this case, the mapping from utterance/sentence to semantic representation is automatically learned from examples of such mappings. In the case of the SLU the corpora of such examples is usually small (in the order of thousands of examples), the examples are domain dependent (i.e. for the task of the dialogue system) and the semantic representation is shallow and ad-hoc to the dialogue systems. On the other hand, SRL corpora is larger (in the tens of thousands), tries to be open domain, and uses semantic roles to build the semantic representation.

Previous approaches for SLU look at the task as a classification task; each element of the semantic representation is decided independently of the other ones. This approach depends on a rich set of features which represent the relations between utterance and semantic representation. Recently, the Hidden Vector State approachHe and Young [HVS, 2005] looked into the SLU task as sequencing task with a tree like structure. This was an important departure from previous approaches because the HVS, besides learning the mappings from the elements of the semantic representation, learns the relations among them; that is it learns the structure of the semantic representation. However, the HVS is a generative approach which makes it harder to extend to a richer set of features as it is achieved by classification approaches. In this direction, Zettlemoyer and Collins [2007] propose a discriminative approach which relies on Combinatorial Categorical Grammar (CCG) to build a semantic representation. Because of its nature it is able to exploit different features between syntactic and semantic elements and the utterances. However, this approach relies on logical forms as the semantic representation which are not common in the development of dialogue systems, as they are harder to label.

On the other hand, SRL approaches tend to split the task in a pipeline fashion. First the basic elements of the task are identified, and later these are related. This approach has the disadvantage that errors propagate in the pipeline, and in order to handle them the system has to carry with n-best list of the previous decisions, which results in complex systems which are harder to maintain.

In this work, we perform semantic processing using the novel approach of Markov Logic [ML, Richardson and Domingos, 2006]. ML is a Statistical Relational Learning language based on First Order Logic (FOL) and Markov Networks (MN). ML separates the modelling of a task from the machine learning process. We use FOL to describe the elements of the semantic representation and the relations between them. These relations are transformed into MNs which are used to create statistical models of the semantic processing task. In the case of the SLU we can take advantage of the MN in

order to represent rich sets of relations between utterances and elements of the semantic representations, but at the same time, using the FOL of the ML framework we can represent relations between the elements of the semantic representations and we can also impose a structure onto it. This is can be done for the ad-hoc and shallow semantic representations common in dialogue systems. Similarly, in the case of the SRL task, we can define a ML model which performs the task in a global fashion and does not follow a pipeline architecture. In the following sections, I present the SLU and SRL task and my proposals for each of them in more detail.

1.1.1 The Spoken Language Understanding task

The main goal of a dialogue system is to hold a conversation with a human user in order to help him achieve his goal. The dialogue system must therefore identify the meaning of the user interaction so as to provide an adequate response. This is modelled in several stages (for instance, automated speech recognition (ASR)), culminating in the phase known as semantic processing. Semantic processing is performed by the Spoken Language Understanding (SLU) module, and involves constructing semantic representations from a sequence of recognised words as produced by the ASR module. The SLU module will identify a semantic representation for this utterance and will pass it to the Dialogue Manager (DM) module. This latter module will interpret it in the context of the current dialogue, and will produce a response. Figure 1.1 presents an example of the type of utterance being processed and the semantic representation created. In this example, the semantic representation is a frame-based representation. It has a main frame identified by the goal of the task and set of slot-values which encode the semantic content of the utterance.

As we will discuss in chapter 2, there are different approaches to performing SLU. Recently, statistical approaches have been used explored for this task [Bod, 2000; He and Young, 2006; Zettlemoyer and Collins, 2007; Wong and Mooney, 2007], rather than the more brittle and labour-intensive grammar-based frameworks. These approaches are data-driven, which means they model the task given a corpus of examples of utterances and semantic representations. However, most of these approaches rely on extra resources, such as gazetteers or extra labellings. This thesis focuses on a case where such resources are not available.

On the other hand, approaches to the SLU task make a trade-off between the complexity of semantic representations and semantic relations they are able to handle.

what flight are there arriving in chicago on continental airlines after n2300

```
GOAL = flight
TOLOC.CITY_NAME=chicago
AIRLINE_NAME=continental airlines
ARRIVE_TIME.TIME_RELATIVE=after
ARRIVE_TIME.TIME=n2300
```

Figure 1.1: Example of semantic processing in a spoken dialogue system. The spoken language understanding module receives an utterance as input and produces the semantic representation as output. In this case, it is a frame-based semantic representation.

Since the semantic representations vary from system to system, dialogue systems developers prefer shallow semantic representations which facilitate the labelling during the development of a dialogue system. However, this situation makes it harder to incorporate the semantic relations between the elements of the semantic representation in a statistical model whilst keeping a single framework. For instance, figure 1.1 we notice that *chicago* is a destination city given its local context (in particular the word *arriving*). This choice influences the choice of slots for the time. From their local context we can infer that the slots for *after 2300* are about time, but together with the slot for *chicago* it is most likely the time is an arrival time than a departure time.

In this work, we propose to use the ML framework to develop SLU modules for a dialogue system. To this end, we treat the semantic processing task as a structured classification task. This means the task is a labelling task with the property that elements of the semantic representation are related to elements of the utterance and other elements of the semantic representation. For instance, for the example in Figure 1.1 we expect to capture the relation between the `ARRIVE_TIME.TIME` slot and the numeral 2300, this is a relation between a semantic element with the utterance. But we also expect to capture the relation of this slot with the `TOLOC.CITY_NAME` slot, this is a relation between a two semantic elements. The ML framework allows us to encode the semantic representation and its relations with First Order Logic (FOL). For instance, we can formulate the first order Markov assumption between the slots of the semantic representation in Figure 1.1. This will account for the relation between the two

time slots. The properties of the ML framework make it easy to develop and experiment with it. Recent developments in the inference method of the ML framework have made the training fast [Riedel, 2008]. This together with the option of extending the model provides an ideal framework for experimentation, allowing easy testing of extensions to the model. The extensions can be made in terms of new information or new relations between the semantic representations. Our goal with this task is to measure the adequacy of the ML framework for SLU. In particular, we focus on using minimal resources, and we test the ML framework in the development of a SLU module.

1.1.2 The Semantic Role Labelling task

In the case of Semantic Role Labelling (SRL), the task is to identify in a written sentence a set of verbal and nominal predicates together with their arguments. Besides linking predicates with arguments, it is necessary to label their semantic roles. Figure 1.2 shows an example of the type of semantic representations produced by the SRL task. The predicate `plays.02`, and the roles for the arguments of it are labelled. With this information we are able to build a semantic representation for the sentence. Note that the SRL semantic representation differs from the frame-based semantic presented in Figure 1.1 in that it allows more than one predicate for sentence and it has scope in the sense that it allows predicates to take as arguments other predicates. In frame-based because the semantic representation is flattened into slot-values there are not predicates but only a frame represented by a goal or subtask label and the slots are only attached to values.

The availability of corpora labelled for the task has made possible the advancement of different techniques for the task [Surdeanu et al., 2008]. In a similar way to our model for SLU we propose to treat the SRL task as structured classification. This allows us the advantage of incorporating in our model correlations between the elements we are looking to identify: for instance, between the predicates and the arguments, or between predicate-argument pairs and their semantic role labels. This is different from traditional approaches to SRL, which divide the task into four stages: predicate identification, argument identification and classification, role classification, and sense disambiguation. In these approaches, it is tricky to incorporate correlation between elements of different stages. Since SRL has a richer semantic representation, this work explores the effect of the different relations we can define between the elements of the task outcome. This gives us a further insight into the advantages and

disadvantages of possible ML models.

Ms. Haag plays Elianti .
Labelling: $[_{ARG0}$*Ms. Haag*$]$ $[_{play.02}$ *plays*$]$ $[_{ARG1}$ *Elianti*$]$
Semantic representation:`plays.02(Ms. Haag,Elianti)`

Figure 1.2: Example of SRL labelling and its semantic representation. The SRL task assigns the roles to the unknown arguments of the predicate `play.02`. With this we are able to render the semantic representation.

We are exploring the SRL task using the ML framework so we can measure the adequacy of the ML model for performing semantic processing with a shallow semantic representation which has a notion of scope. The notion of scope is usually left out of shallow semantic representations. With this exploration we aim to investigate if the ML framework can be used in future NLP tasks which require scope in their semantic representations.

1.2 Semantic representations

As we have explained, the choice of semantic representation is an important factor for semantic processing. In practical dialogue systems this is defined by the capabilities of the dialogue manager, and its ability to handle the corresponding semantic representation. Although a richer semantic representation would lead to a better representation for the interaction of the user, this has not been the choice for practical dialogue systems where shallow semantic representations are currently preferred. Frame-based semantic representations are common within the community [Seneff et al., 1999; Bennacef et al., 1996; Gupta et al., 2006; Varges and Purver, 2006; Bonnema et al., 1998; Lemon et al., 2006a]. Frame-based semantic representations have the advantage of being easy to label, whilst still being informative enough to build a dialogue system around them. An example of a frame-based semantic representation is shown in Figure 1.1.

The origins of the frame-based semantic representation can be traced to Minsky [1974]. However, in recent systems the frame-based semantic representation has been treated as a data structure, an *attribute-value structure*, which encodes the semantic content of the utterance. In particular, there are two types of content: *dialogue act*/*goal*

Dialogue acts	Goals
REQUEST-INFO	And, what city are you flying to?
PRESENT-INFO	The airfare for this trip is 390 dollars.
OFFER	Would you like me to hold this option?
ACKNOWLEDGMENT	I will book this leg.
STATUS-REPORT	Accessing the database; this might take a few seconds.
EXPLICITCONFIRM	You will depart on September 1st. Is that correct?
IMPLICIT-CONFIRM	Leaving from Dallas.
INSTRUCTION	Try saying a short sentence.
APOLOGY	Sorry, I didn't understand that.
OPENINGS/CLOSINGS	Hello. Welcome to the C M U Communicator.

Table 1.1: Dialogue act examples taken from Walker and Passonneau [2001].

and *slot-values*. In the case of dialogue acts and/or goals, the semantic representation is treated as a property of the whole utterance. In the case of the slot-values, the semantic representation is treated as a property of the words, and it is the whole set of slot-values which defines the meaning of the utterance. In the following section, we describe in detail each one of these elements.

1.2.1 Dialogue acts and goals

Dialogue acts are linguistic representations of the communicative intent of a user or system utterance [Walker and Passonneau, 2001; Frampton and Lemon, 2008]. The intent is represented by a functional category, for instance *REQUEST* is the dialogue act for an utterance like *can you tell me the time?*. The dialogue act aims to capture the functional similarity between different utterance modalities. For instance the imperative utterance *please tell me the time* shares the same interrogative dialogue act as *can you tell me the time?*. There are different taxonomies for dialogue acts which are used in dialogue systems. For example, Table 1.1 presents the taxonomy proposed in Dialogue Act Tagging Scheme for Evaluation [DATE, Walker and Passonneau, 2001]. In the development of a dialogue system, the system developer chooses which set of dialogue acts to use; it can use a reduced set or an extended set with domain dependent categories.

Task oriented spoken dialogue systems also represent the *goal*. The goal is a semantic category which identifies the task the dialogue system has to perform for that

given utterance. In Figure 1.1 the goal of the utterance is `FLIGHT`: this means that the task that has to be performed relates to booking a flight. The *goal* is a domain and system-dependent semantic representation.

In summary, the different options for dialogue acts and goals make it difficult to create a unique system and domain independent semantic parser for different dialogue systems. Our data driven approach reduces this problem, since it depends on examples of the semantic representation. This means it only models the dialogue acts and goal which are present in the examples.

1.2.2 Slot-values

The slot-values are used to describe the semantic content of the utterance. The slots describe attributes which are relevant for the dialogue manager, so that it can perform the task requested by the user. Once the dialogue manager has enough values for these attributes it will perform the task, or if there are too many values missing, it can ask for the specific attributes needed. In Figure 1.1 we can see four slots, which describe three aspects of the task: the destination, the airline name and the arrival time. The slots are composed of semantic concepts: for instance, `ARRIVE_TIME.RELATIVE_TIME` has two concepts separated by a dot. Slot-values are domain and task dependent and the definition of the slots and possible values for the dialogue system are left to the developer.

The frame-based corpora that we use in this work has a slot per each value in the utterance. We call this *plain slot-values*, since there is not an explicit structure between the slots. However, plain slot-values have a simple notion of compositional semantics. For instance, in the case of the arrival time in the example, this is defined by two slots (i.e., `ARRIVE_TIME.RELATIVE_TIME`, and `ARRIVE_TIME.TIME`). When using slot-values, the dialogue manager has to define a way of interpreting them. For instance, in the example, in order to find the right arrival time, the dialogue manager has to incorporate a rule which says that two consecutive `ARRIVE_TIME` slots ought to be interpreted together. With this, the two values are put together to represent the right time. This is a general problem with the plain version of slot-value representations which does not incorporate a notion of scope or hierarchy among the semantic concepts and slots. There are some proposals for incorporating richer structural information in the slot-values, the so-called *richer slot-values* (e.g., [Veldhuijzen van Zanten, 1996] proposes scoping for slots, and [He and Young, 2005] proposes an extra labelling for

representing the tree structure relations of the slots). However, given that we want to compare this system with state-of-the-art approaches to semantic processing while using the less resources as possible, we will keep working with the shallow version of the slot-values.

1.2.3 Semantic roles

Semantic Role Labelling adds a layer of predicate-argument information, or semantic roles, to syntactic structures of the Penn Treebank. The resulting semantic representations can be thought of as shallow, in that they do not represent co-reference (in the sense used in Computational Linguistics), quantification, or many other higher-order phenomena. However, they are broad in the sense that they cover every instance of verbs and nominals in the corpus [Palmer et al., 2005]. Figure 1.2 presents an example of semantic role labelling[1]. In this case, the predicate is *play*, which is related to the word *plays*. This predicate has two arguments: *Haag* and *Elianti* with the semantic roles *ARG*0 and *ARG*1. These semantic roles are related to the frame of the predicate, which is *play*, and its sense 02. If we look for it in the Propbank frame sets, we found that this sense corresponds to the meaning of "playing a role". In this case, *ARG*0 defines the actor and *ARG*1 defines the role. The goal of SRL is to build these semantic representations for verbal and nominal predicates.

1.3 Evaluation metrics

Two metrics have been proposed for measuring the performance of a SLU module. He and Young [2005] proposed a metric which we will refer to as *global*, which measures the capability of an SLU module to identify slot-values for a set of utterances. There are two versions of this metric. The first one, used by He and Young [2005] does not include the goal slot as a part of the slot-values. In addition, He and Young [2005] propose to measure the accuracy of the dialogue act and/or goal. The second version of the global metric is the one used by Zettlemoyer and Collins [2007], which includes the goal as another slot-value. In addition, Zettlemoyer and Collins [2007] proposed another metric which measures the capability of an SLU to recover correct frames per utterance. In this work, we also propose a third metric, which we will refer to as *average*; this metric measures the capability of a SLU module to recover slot-values

[1] From CONLL-05 Shared-task corpora [Carreras and Márquez, 2005].

per utterance. The three metrics calculate precision and recall in their own terms, and the f-score based on their precision and recall . The formulae for each metric are as follows:

$$
\begin{aligned}
Global: \quad Precision &= \frac{total\ \#\ true\ positives}{total\ \#\ true\ positives + total\ \#\ false\ positives} \\
Recall &= \frac{total\ \#\ true\ positives}{total\ \#\ true\ positives + total\ \#\ false\ negatives} \\
Average: \quad Precision &= \frac{\sum_{u \in\ utterances} \frac{\#\ of\ true\ positives\ in\ u}{\#\ of\ true\ positves\ in\ u + \#\ of\ false\ positives\ in\ u}}{Total\ \#\ utterances} \\
Recall &= \frac{\sum_{u \in utterances} \frac{\#\ of\ true\ positives\ in\ u}{\#\ of\ true\ positves\ in\ u + \#\ of\ false\ negatives\ in\ u}}{Total\ \#\ utterances} \\
Exactmatch: \quad Precision &= \frac{\#\ correct\ analysis}{\#\ parsed\ utterances} \\
Recall &= \frac{\#\ correct\ analysis}{Total \#\ utterances}
\end{aligned}
$$

The formula for the f-scores is:

$$f - score = \frac{2 \times precision \times recall}{precision + recall}$$

To clarify the differences between these three metrics we use the synthetic example in Table 1.2. For the *global* metric, we use the sum of true positives, false positives, and false negatives without regard to which utterance they belong:

$$
\begin{aligned}
Precision &= \frac{3+3+2}{3+3+2+2+1} \\
&= 0.7272 \\
Recall &= \frac{3+3+2}{3+3+2+1+3} \\
&= 0.6666
\end{aligned}
$$

In the case of the *average* metric, we calculate the precision and recall of each independent utterance, and then average these:

$$
\begin{aligned}
Precision &= \frac{\frac{3}{3+2} + \frac{0}{0+1} + \frac{3}{3+1} + \frac{2}{2+0}}{4} \\
&= 0.5875 \\
Recall &= \frac{\frac{3}{3+1} + \frac{0}{0+3} + \frac{3}{3+0} + \frac{2}{2+0}}{4} \\
&= 0.6875
\end{aligned}
$$

Utterance	Total slots	True positives	False positives	False Negatives
1	4	3	2	1
2	3	0	1	3
3	3	3	1	0
4	2	2	0	0

Table 1.2: Synthetic slot-labelling example.

Finally, for the *exact match* metric, we only consider the cases where the semantic representation was exactly as expected, and the cases where we generate a semantic representation:

$$\begin{aligned} \textit{Precision} &= \frac{1}{4} \\ &= 0.25 \\ \textit{Recall} &= \frac{1}{4} \\ &= 0.25 \end{aligned}$$

Recently SRL systems have been evaluated in terms of precision, recall and f-scores of the identified semantic dependencies. A semantic dependency corresponds to the tuple of the positions of a predicate and one of its arguments plus the semantic role labelling. In the case of predicates, there is a special semantic dependency which links them with an artificial *ROOT* position, and therefore the evaluation takes into account the predicate identification. For our example of Figure 1.2 the resulting semantic dependencies are:

$$\begin{aligned} \textit{ROOT}, 3, \textit{play}.02 \\ 3, 2, \textit{ARG}0 \\ 3, 4, \textit{ARG}1 \end{aligned}$$

In evaluating a system there are two ways in which its performance is measured:

- *Unlabelled* performance measures the accuracy of the system in identifying the links of the semantic dependencies. In this case, we only consider to be correct the dependencies those positions match the gold standard reference.

- *Labelled* performance measures the accuracy of assigning semantic roles to the dependencies that have been correctly identified. In this case, besides getting the positions right, the semantic role labels have to match the reference.

There are some variants that give some partial scores to partial positions of the arguments. However, in this work we consider an argument to be correct when the boundaries perfectly match the argument in the reference solution.

1.4 Contributions

This thesis focuses on the application of Markov Logic for the SLU and SRL semantic processing tasks. The contributions of this work are:

- We perform the SLU task with limited resources. Most current approaches rely on additional resources, such gazetteers and additional labellings, in order to perform the task. In our case, we explore the adequacy of the Markov Logic [ML, Richardson and Domingos, 2006] for SLU with minimal resources. ML distinguishes between local and global dependencies. We hypothesised that the inclusion of global dependencies would help this system to equal the results provided by state-of-the-art models, without using extra resources (see Chapter 5 and Chapter 6).

- We explore the ML framework by using more complex shallow semantic representations than those commonly used in dialogue systems. Since ML is a relatively new framework, there is not enough evidence that it could generalise with more complex semantic representations: in particular, for shallow representations with a sense of scope. For this reason, we also tested the framework for Semantic Role Labelling. Within this setup we created a joint model of the task and we experimented with different versions of our model which give us an insight into the types of structured relations that are most beneficial for the ML approach (see Chapter 7).

- Additionally, we test a semantic module developed with the ML approach for a state-of-the-art dialogue system. The development setup of this module is similar to the goal we tested in this work: there were no extra resources, the semantic representation was shallow and tailored to the system, but we were able to incorporate long distance dependencies in the final model: for instance,

to capture the correlation between the price and food type slots in utterances such as: *A cheap place to eat pizza* (see Chapter 8).

These three contributions point out the adequacy of the Markov Logic approach for designing state-of-the-art Spoken Language Understanding modules for dialogue systems, and their adequacy for other Natural Language Processing tasks involving a semantic representations, such as Semantic Role Labelling.

1.5 Outline of the thesis

Chapter 2 presents the previous work done on semantic processing. It also presents the more recent results from the approaches which are relevant to our work.

Chapter 3 presents the different corpora we use in this work, and some of their statistical properties.

Chapter 4 describes the main data-driven frameworks we use: Hidden Vector State [HVS, He and Young, 2005] and Markov Logic and [ML, Richardson and Domingos, 2006].

Chapter 5 describes our experiments which measure the adequacy of ML for performing SLU for dialogue systems. In this chapter, we introduce a set of baseline results to which we compare our first ML model; these were published as Meza-Ruiz et al. [2008a].

Chapter 6 describes our experiments and presents our results along with the modified architecture of the ML model. A previous version of these results were published in Meza-Ruiz et al. [2008b].

Chapter 7 describes the experiments and present the results of applying ML to perform Semantic Role Labelling [Riedel and Meza-Ruiz, 2008; Meza-Ruiz and Riedel, 2009].

Chapter 8 presents our experience and results when we use a ML model in a state-of-the-art dialogue system: TownInfo. In this scenario we also test the performance of the ML model with Automatic Speech Transcriptions.

Chapter 9 summarises the contributions of this work and presents future work.

Chapter 2

Previous work in semantic processing

In this chapter I review previous work on Spoken Language Understanding (SLU) and Semantic Role Labelling (SRL). For the case of the SLU approaches, I introduce the following seven aspects which will allow me to characterise each of the approaches:

1. The robustness of the approach. This is the ability of the semantic module to cope with ungrammatical input with regard to the wording of the utterances. This is an important feature, since many utterances in spoken dialogue systems are ungrammatical, have disfluencies and/or unknown words.

2. The source of knowledge used in the approach. There are knowledge-based approaches which rely on the expertise of the system developer to hand-craft the semantic module, or data-driven approaches which use examples to model the task of the semantic module in a statistical framework.

3. Different computational frameworks. The main ones are: parsing, classification, and sequencing tasks.

4. Whether syntactic analysis is part of the approach. Since semantic processing covers from a word string to a semantic representation, it is possible to include syntactic analysis as part of the semantic processing. The approaches vary from not doing syntactic analysis to deriving the semantic representation from syntactic trees.

5. The semantic representation used. The approaches presented here use different semantic representations, the slot-values representation being the most common one. In section 1.2 we presented a more detailed discussion about the semantic representations mentioned here.

6. Whether the semantic module is portable among other domains.

7. Whether it affords a straightforward interface to other modules, in particular to the Dialogue Manager (DM). In most of the cases the connectivity with the dialogue manager is straightforward since this defines the type of semantic representation required. However, sometimes when off-the-shelf resources are used it is necessary to translate their semantic representation output to the one required by the DM.

For each approach we present in this chapter we describe each of these aspects, so we can compare between them.

For the case of SRL I introduce the following aspects:

- Experimental setting. The SRL task has two experimental settings. In the first setting, the predicates and their senses are known. In the second setting these elements are unknown.

- Syntactic information. The SRL task relies on syntactic information. The task has been defined for the constituent-based and dependency-based parses.

- Architecture of the approach. The SRL task involves several decision stages. The architecture of the approach connects such decisions in different ways. It can be done in a pipeline fashion where the output of one is the input of the next one, or it can perform in a joint fashion where all decisions are made at the same time.

- Features. Most of the approaches map the linguistic knowledge into features. The sets vary between the approaches.

- Corpora. The corpus which was used to create the SRL defines the language and the type of predicates. Most of the research is focused on English but there are corpora available for Catalan, Chinese, Czech, German, Japanese and Spanish. There are two types of predicates: verbal for verbal phrases, and nominals for nominal phrases.

This chapter is divided into three parts. First, section 2.1 presents the knowledge-based approaches. Second, section 2.2 presents the data-driven approaches. In order to exemplify the different semantic representations in these two parts we use the utterance *i want a flight to chicago from dallas*. Finally, the third part presents a summary of the work in SRL related to our implementation of an SRL system.

2.1 Knowledge-based approaches

In a knowledge-based approach, the dialogue system developer designs and implements a semantic parsing module. The system developer has to come up with lexical or syntactic patterns from which to build a semantic representation. Anticipating all possible expressions of a dialogue system is not a simple task and typically these approaches therefore lack robustness. Recently, there has been interest in the community in overcoming this problem. Some of these work is presented in the following sections.

The simplest approach among the knowledge-based approaches is *word spotting*, which consists of keywords and/or lexical patterns. The next level is to hand-craft a grammar, which consists of syntactic patterns. To overcome the problem of writing out all possible patterns some dialogue systems adapt already hand-crafted grammars with the property that they were wide coverage: that is they are not designed for a particular domain, and generally include all sorts of syntactic phenomena. The following sections present each of these approaches in detail.

2.1.1 Word spotting

In this approach, a set of keywords, lexical patterns, or finite state machines are proposed. The goal of these is to identify key elements in the utterance, and they are popular in current state-of-the-art dialogue systems [Gorin et al., 1997; Lemon et al., 2006b]. This is because the technique, although simple, is quite robust for spoken language in a dialogue system and is relatively easy to implement. However, its main disadvantage is that the generated semantic representations are minimal (i.e., keywords): they only consist of values or named entities. To overcome this problem usually involves making the patterns more complex in order to incorporate long distance dependencies. Dialogue systems usually include word spotting at a prepossessing stage.

Robust	:	Yes
Source of knowledge	:	System developer
Nature of task	:	Pattern matching
Syntactic analysis	:	No
Semantic representation	:	Set of keywords, example: $\{origin = chicago, destination = dallas\}$
Portability	:	Need to rewrite the keywords, patterns or finite state machines
Connectivity with DM	:	Straightforward for DMs requiring keywords.

2.1.2 Hand-crafted grammars

In this type of approach system developers have to design syntactic grammars to parse the utterances [Allen et al., 1996; Dzikovska et al., 2005; Lemon et al., 2001]. The grammars rely on syntactic information and rules in order to build a syntactic tree from which the semantic representation is instantiated. However, designing syntactic patterns (i.e., the rules of a grammar) is not trivial. This situation results in grammars that are not robust and/or are ambiguous. This means the dialogue system has to deal with cases where nothing is parsed or where there is more than one parse. However, an advantage of this approach is the different semantic representations that can be produced from the syntactic trees, which depending on the effort put into the implementation of the grammar can include different semantic phenomena.

Robust	:	No
Source of knowledge	:	System developer
Nature of task	:	Parsing
Syntactic analysis	:	Yes
Semantic representation	:	Potentially Logical forms, example: $want(user, flight(origin(city(chicago)), destination(city(dallas))))$
Portability	:	Necessary to rewrite a new grammar
Connectivity with DM	:	Straightforward.

For example, Ranta [2002] presents the Grammatical Framework (GF) framework for developing grammars which ease the design of handcrafted grammars. GF separates syntax and semantics. Then, the system developer can focus on the design of the semantic level and later focus on the syntactic level. The system developer can even

design the syntactic level for different languages while using the same design of the semantic level. Additionally, GF tackles the problem of robustness by allowing deletions of unknown words. However, GF still requires an expert to design the grammar, in particular for the syntactic level. In addition, it can not deal with unknown words that are relevant to the semantic representation since they are not considered during parsing.

2.1.3 Wide-coverage grammars

In this type of approach an off-the-shelf wide-coverage grammar is used (e.g., [Purver, 2002]). These are hand-crafted grammars developed for broad coverage of language rather than being domain specific. Although they are an attractive option for semantic parsing, they are not very common in dialogue systems given that they lead to several semantic representations, i.e., they are ambiguous. This is because by design they are domain neutral (e.g., LINGO project [Oepen et al., 2002]). Besides, since they were not designed for spoken language they are not robust enough for it and in most cases some work is necessary to generate the semantic representation needed by the dialogue system.

Robust	:	No
Source of knowledge	:	Wide coverage grammar developers
Nature of task	:	Parsing
Syntactic analysis	:	Yes
Semantic representation	:	Minimal Recursion Semantics, Hybrid Logic, example: $\{h_1 : want(x,y), h_2 : user(x), h_3 : flight(y), h_4 : origin(x, chicago : city), h_5 : destinion(x, dallas : city)\}$
Portability	:	Portable to any system
Connectivity with DM	:	Necessary to adapt the output to adequate semantic representation.

2.2 Data-driven approaches

Data-driven approaches rely on a corpus in order to model semantic processing as a stochastic process. These approaches tend to be more robust than the knowledge-based

ones since they model the relations between utterances and semantic representation directly from the examples of the corpus. This advantage is at the same time a disadvantage, because these approaches require a labelled corpus. The labelling of a corpus is a costly task in dialogue system development. Some of the approaches go further and require extra sources (e.g., gazetteers). In this work, we examine the case where the cost of the development of the semantic module is reduced by using an easy to annotate semantic representation, where no extra sources are required, and we use off-the-shelf resources.

2.2.1 Divide-and-conquer approach

A common approach for SLU for dialogue systems is a divide-and-conquer approach where the task is divided into subtasks. Basically, one subtask focuses on the semantic representations attached at the word level (e.g., slot-values) while another subtask focuses on the utterance level (e.g., speech act, goal, subtask). Later both results are put together to form the final semantic representation. Dialogue systems model the subtasks using different methods. The most common subtasks are:

1. Named Entity Recognition (NER) for the words in the utterances. This is used to identify the classes of the values (e.g., `chicago` is labelled as `city`) [Gupta et al., 2006].

2. Classification of the words of a utterance into slots (e.g., `chicago` as `destination.city`). Common methods include: Maximum entropy [Varges and Purver, 2006] and Support Vector Machines [Wu et al., 2006; Haffner et al., 2003].

3. Sequencing the slots of an utterance. Common methods include: n-grams [Weilhammer et al., 2006] and Conditional Random Fields.

4. Classification of goal or dialogue act. Common methods include: Maximum entropy, boosting [Gupta et al., 2006; Fabbrizio et al., 2002], Support Vector Machines [Ji and Bilmes, 2005].

A disadvantage of this approach is that the tasks are linked with a pipe-line architecture, where the output of a subtask is the input to the next one. This means there is no explicit way to model dependencies in other direction of the pipe-line, for example,

the dependency of the goal with the semantic representation at the word level.

Robust	:	Yes
Source of knowledge	:	Corpus annotated, plus features analysis
Nature of task	:	Combination of subtasks
Syntactic analysis	:	No
Semantic representation	:	Plain slot-values, example: $\{city = chicago, city = dallas\}$ or $\{origin.city = chicago, destination.city = dallas\}$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward for slot-values DM.

2.2.2 Data-Oriented Parsing

Bod [2000] presents a parsing framework for dialogue systems using Data-Oriented Parsing (DOP). In this case, the DOP framework is used to learn a grammar from syntactic trees which are semantically augmented. The resulting semantic representation consists of slot-values based on the definition proposed by Veldhuijzen van Zanten [1996]. For their experiments they used the OVIS corpus [Bonnema et al., 1998] and measured exact matches. Their results are presented in Table 2.1. However, to achieve

Precision	85.8%
Recall	86.9%
Match	80.6%

Table 2.1: Best DOP parser results for OVIS corpus

these results the developed system is reported to be slow for a practical dialogue system (it takes minutes to parse a sentence). A faster version of the system reports less good results (see Table 2.2). A disadvantage of this work is that it requires a corpus

Precision	83.0%
Recall	84.3%
Match	78.8%

Table 2.2: Faster DOP parser for OVIS corpus

with both syntactic and semantic annotations.

Robust	:	Yes
Source of knowledge	:	Semantic and Syntax annotated corpora
Nature of task	:	Parse selection
Sytactic analysis	:	Yes
Semantic representation	:	Richer slot-values with structure, example: $flight.(origin.city = chicago, destination.city = dallas)$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward for slot-value DMs.

2.2.3 Hidden Vector State approach

He and Young [2005] proposed to use the Hidden Vector State (HVS) for semantic processing. This approach models the semantic task as a sequencing task: that is, it models the sequence of slot-values for an utterance. We can think of the HVS model in terms of an extended Hidden Markov Model (HMM) which, instead of having a unique state for each word in the utterance, has a vector of states. Together with this modification the HVS model proposes that the vector states behave as elements in a stack. This allows it to define the transition probability from one vector state to another in terms of the probability of the pop and push stack operations. We present the HVS approach details in the Chapter 4, where we present our experiments using this approach. Table 2.3 shows the performance of the approach with respect to identifying slot-value pairs.

Precision	89.82%
Recall	88.79%
f-score	89.28%

Table 2.3: HVS results for ATIS corpus

A disadvantage of this method is that HVS has proved not to adapt well to new tasks. In Weilhammer et al. [2006] an $n-gram$ approach is reported to have a better performance. Our own results with the HVS approach are mixed, with good results for some corpora and less good for others.

Robust	:	Yes
Source of knowledge	:	Corpus with slot-values, abstract annotations and gazetteer
Nature of task	:	Sequencing
Syntactic analysis	:	No
Semantic representation	:	Plain slot-values, example: $\{goal = flight, origin.city = chicago, destination.city = dallas\}$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward for slot-value DMs.

2.2.4 Clustering

Ye and Young [2006] present an algorithm for SLU. The proposed approach uses an unsupervised sentence clustering technique called Y-clustering to automatically select a set of exemplar sentences from a training corpus. These exemplars are combined with simple sentence-level semantic annotations to form templates which are then used for semantic decoding. The performance of the approach is presented in Table 2.4.

Precision	93.75%
Recall	93.09%
f-score	93.42%

Table 2.4: Clustering results for ATIS corpus

Robust	:	Yes
Source of knowledge	:	Partial corpus with slot-values and gazetteer
Nature of task	:	Sequencing
Syntactic analysis	:	No
Semantic representation	:	Plain slot-values, example: $\{origin.city = chicago, destination.city = dallas\}$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward for slot-value DMs.

A disadvantage of this approach is that for now it has only been defined for slot-values and we do not know how good will perform with other semantic representations. Another disadvantage is that the approach depends on a gazetteer.

2.2.5 Relaxed CCG grammar

Zettlemoyer and Collins [2007] propose an online method to learn logical forms by learning a combinatorial categorical grammar (CCG). For this, they introduce non-standard CCG combinators that relax certain parts of the grammar – for example allowing flexible word order, or insertion of lexical items – with a learnt cost.

This work is important to us, because it applies a global method to the task and in addition it used the ATIS corpus. So far, their results are the best for the corpus. The performance of the approach is presented in Table 2.5. These results were obtained with the $NOV93$ test set of ATIS 3 corpus. A disadvantage of the proposed method is

Precision	95.11%
Recall	96.71%
f-score	95.9%

Table 2.5: CCG results for ATIS-$NOV93$ corpus

that it requires logical forms for training. Although logical forms are a richer semantic representation, they are expensive to label during the development of a dialogue system.

Robustness	:	Yes
Source of knowledge	:	Corpus with logical forms with gazetteer, plus features analysis
Nature of task	:	Parsing
Syntactic analysis	:	Yes
Semantic representation	:	Logical forms, example: $\lambda x.(flight(x) \wedge origin(x, chicago : city) \wedge destination(x, dallas : city))$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward.

2.3 Semantic Role Labelling

There has been a great amount of work on Semantic Role Labelling in recent years. In this section, we present previous approaches to SRL which are related to our approach.

2.3.1 Features for SRL

Xue and Palmer [2004] present an analysis of the features traditionally used in the SRL task. They also suggest a new set of syntactic features which in their experiment made a difference in the performance. In our SRL systems we use a similar group of features to those proposed in this work. In the following list, we enumerate the features used by this system. We mark the features which were new to the task.

Experimental setting	:	It knows which are the predicates. It ignores the frames and senses. It has to identify the arguments and tag them with their semantic role.
Syntactic information	:	Constituent-based.
Architecture	:	Pipeline: 1. Constituent pruning. 2. Argument identification. 3. Argument classification.
Features	:	Predicate, path, constituent type, position, voice, head word, sub-categorisation. The new features proposed are: Syntactic frame, lexicalised constituent type, lexicalised head word, position of voice, head of PP
Corpora	:	Propbank.

2.3.2 Joint model

Toutanova et al. [2005] present a joint model for SRL. In this approach, a set of global features are proposed. These features have the goal of representing the relations between arguments and their semantic roles. This is not easy since this is the information we are looking for. In order to incorporate this information, this approach presents a re-ranking setting, where the n-best assignation of semantic roles are re-ranked. The following table presents a summary of the approach.

Experimental setting	:	It knows which are the predicates. It ignores and senses. It has to identify the arguments and tag them with their semantic role.
Syntactic information	:	Constituent-based.
Architecture	:	Pipeline and re-ranker: 1. Constituent pruning. 2. Argument identification. 3. Argument classification. 4. Re-rank *n*-best SRL
Features	:	Local and global features. The global features are: Label sequences and Syntactic frame for labelling (includes lexicalised version of the features).
Corpora	:	Propbank.

2.3.3 Global semantic model

Johansson and Nugues [2008] present a similar approach to Toutanova et al. [2005]. This work is important to us because it is ranked as the best SRL in the Closed track of the CoNLL-08 Shared task. This is also the system reported to have the best performance in the PropBank corpus. For some of our experiments we share a similar setup to that presented in this work.

Experimental setting	:	It performs dependency parsing and SRL. For the SRL task it predicts predicates, and their senses, identifies arguments for such predicates and tags them with their semantic role.
Syntactic information	:	Dependency-based.
Architecture	:	Pipeline and re-ranker: 1. Constituent pruning. 2. Argument identification. 3. Argument classification. 4. Linguistic constraint and Re-rank *n*-best predicate argument pairs. 5. Syntactic-semantic re-ranker
Features	:	Local and global features.
Corpora	:	Propbank and NomBank.

2.3.4 Sentence simplification

Vickrey and Koller [2008] present a novel approach to the SRL task. They use sentence simplification to model the task. This requires the generation of a set of simplified sentences from the sentences in the corpus. A set of handwritten rules are specified to simplify the sentences. The set of simplified sentences are used to train a log-linear

model for the SRL labellings. This work is relevant to us because is ranked as the best SRL system in the Open track of the CoNLL-08 Shared task.

Experimental setting	:	It predicts predicates and their senses, identifiifies arguments for such predicates and tags them with their semantic role.
Syntactic information	:	Dependency-based.
Architecture	:	Simplification module and Pipeline: 1. Identify potential predicates. 2. Label roles 3. Eliminate predicates without arguments. 4. Label sentences.
Features	:	Local features.
Corpora	:	PropBank or PropBank and NomBank.

In this approach the system developer has to propose a set of rules for sentence simplification.

2.4 Summary and discussion

In this chapter, we have presented the main approaches to Spoken Language Understanding (SLU) in dialogue systems and the recent developments in Semantic Role Labelling (SRL).

In this work we will focus on the Markov Logic (ML) framework's ability to perform semantic processing [Richardson and Domingos, 2006]. This is a discriminative data-driven approach which models the SLU task as a labelling task. As a source of knowledge it requires a semantically annotated corpus. To minimise the labeling cost of such a corpus we limited ourselves to *slot-values*. The ML framework does not require syntactic analysis. However, it is possible to incorporate this from an off-the-shelf component as a part of the feature analysis. These features are not exclusively limited to the syntax level but can be about any other level (e.g., lexical, semantic). We present a more detailed explanation of the ML framework in Chapter 4. A a summary of our approach to the SLU task is presented next:

Robustness	:	Yes
Source of knowledge	:	Semantic annotated corpus, plus modelling analysis
Nature of task	:	Structured labelling
Syntactic analysis	:	As features
Semantic representation	:	Plain slot-values, Semantic Role Labels, example: $\{goal = flight, origin.city = chicago, destination.city = dallas\}$
Portability	:	Necessary to train module with corpora in domain.
Connectivity with DM	:	Straightforward.

With the ML framework we can perform SRL as a joint task where decisions about the predicate, arguments and roles influence each other. This idea has been explored with good results. We will compare the performance of our model with state-of-the-art approaches in a similar setting. The summary of our approach for the SRL task is presented next:

Experimental setting	:	It predicts predicates and their senses, identifies arguments for such predicates and tags them with their semantic role.
Syntactic information	:	Dependency-based.
Architecture	:	Joint model
Features	:	Local and Global features.
Corpora	:	PropBank or PropBank and NomBank.

In the next chapter we will present a description of the corpora which is used in this work. Chapter 4 will present the main theoretical frameworks of this work, Hidden Vector State [He and Young, 2005] and Markov Logic [ML, **?**]. Chapters 5, 6 and 8 will present different aspects for the SLU task. Chapter 7 will present our system for the SRL task.

Chapter 3

Corpora

This chapter presents the corpora we use in this work. They all consist of utterances or sentences paired with their semantic representations. We use two types of corpora: spoken language based and written language based. Both types are labelled with shallow semantic representations. The first type of corpus were collected in the context of dialogue system research. They are domain/task dependent and relatively small. The corpora of this type that we will use are: the Air Travel Information System corpus [ATIS Dahl and et al., 1994], Communicator corpus [Bennett and Rudnicky, 2002], and TownInfo [Georgila et al., 2005a]. For ATIS and Communicator we use the versions developed by He and Young [2005]. The second type of corpus was collected in the context of Semantic Role Labelling (SRL) research. These corpora are based on the Penn TreeBank [Marcus et al., 1993] and consist of sentences about finance news paired with their semantic roles. These corpora are: PropBank [Palmer et al., 2005] and NomBank Meyers et al. [2004]. These corpora are complementary since they label different types of semantic predicates (verbal and nominal). In our experiments, we will use the union of both corpora provided in the CoNLL-08 Shared Task [Surdeanu et al., 2008].

The corpora here presented will be used to test the adequacy of Markov Logic to perform semantic processing. We will use the ATIS and Communicator corpora to experiment with different models and compare their performance with state-of-the- art approaches (see chapter 5 and chapter 6). We use the PropBank and NomBank corpora to test the adequacy of the Markov Logic to deal with more complex but still shallow semantic representations (see chapter 7). Finally, we use the TownInfo corpus to test Markov Logic for the development of a Spoken Language Understanding module. In this case, there is little labelled data, and the dialogue system has tailored semantic

representations (see Chapter 8).

The outline of the chapter is as follows. Section 3.1 presents the ATIS corpus. Section 3.2 introduces the Communicator corpus. Section 3.3 presents the TownInfo corpus. Each section presents a syntactic and statistical analysis of the respective corpora. Section 3.4 introduces the NomBank and PropBank. Finally, Section 3.5 presents a summary of the properties of the corpora here presented.

3.1 ATIS Corpus

The Air Travel Information System [ATIS, Dahl and et al., 1994] corpus is a collection of spoken dialogues between a human user and a dialogue system in the domain of flight booking. The users obtain air travel information from a relational database, containing information such as flight/ground transportation schedules and fares using spoken natural language. The user uses this information to solve air travel planning scenarios. Some examples of the utterances we find in the corpus are presented in Figure 3.1.

1. *what flight are there arriving in chicago on continental airlines after n2300*
2. *i'll need a rental car at the atlanta airport can you show me what's available*
3. *what does co mean*

Figure 3.1: Example of utterances from ATIS-3 corpus Dahl and et al. [1994].

We will use the examples in the ATIS corpus to learn the mapping from utterances to a semantic representation and to evaluate this process. For this purpose, the ATIS corpus was divided into four parts: training, development, and two testing parts (*NOV*93 and *DEC*94). With the training part we created a model of the mapping of utterances to semantic representations. We tune the parameters of the model using the development part (e.g., number of iterations, loss function, Gaussian priors). Finally, once we had chosen the parameters to create a model we evaluated it on the testing parts.

Table 3.1 presents the number of utterances of the different parts of the corpus. The average length of the utterances in the training part is of 11.21 words, while the median is 12 words (see Figure A.1 for the histogram of the lengths of the training part). The utterances use a total of $50,250$ words with a vocabulary of size of 897 types of words.

Corpus	Number of utterances
Training	4,481
Development	497
Testing *NOV*93	448
Testing *DEC*94	445

Table 3.1: Number of utterances in parts of ATIS corpus.

We will use the ATIS corpus to test the adequacy of Markov Logic for modelling the semantic processing task as defined in He and Young [2006]. In particular, the results we obtained with this corpus are directly comparable with previous approaches to the task [He and Young, 2005, 2006; Zettlemoyer and Collins, 2007] (see Chapter 5). Additionally, we will use this corpus to test some extensions on our model which can be seen in Chapter 6.

3.1.1 Semantic representation

In this work, we use the version of the corpus developed by He and Young [2005]. This version is composed of transcriptions of the utterances and their frame-based semantic representations. In this case, the corpus is labelled with *goals* and *slot-values* (see section 1.2). This labelling was semi-automatic generated from the original SQL-style labelling. Figure 3.2 presents the frame-based labelling for the previous examples shown in Figure 3.1. Each one of the utterances encodes a goal and a set of slot-values. The slots can be composed of one or two semantic concepts; where they have two concepts, these are separated by a period. For instance `TOLOC.CITY_NAME` has two concepts: `TOLOC`, which signifies that the value is a destination, and `CITY_NAME` which signifies that the value is the name of a city. The values of the slots are sequences of words which appear in the utterance. Some of the slots are ad-hoc, for instance in the last example in Figure 3.2, the slot-value `MEANING=mean` seems to be redundant. These slots ensure that frame-based semantic representations can be translated into SQL queries used for evaluating the ATIS corpus [Pallett et al., 1994].

Additionally to the labelling, the corpus contains a set of lexical classes. This is a gazetteer which defines a set of classes to which some words belong. For instance, the words *chicago* and *atlanta* belong to the class *city*. In total there are 31 classes (see Table A.3 for the whole list) which were extracted from the ATIS-3 Database and the trainning corpus. This gazetteer has played an important role in previous approaches

what flight are there arriving in chicago on continental airlines after n2300

```
GOAL = flight
TOLOC.CITY_NAME=chicago
AIRLINE_NAME=continental airlines
ARRIVE_TIME.TIME_RELATIVE=after
ARRIVE_TIME.TIME=n2300
```

i'll need a rental car at the atlanta airport can you show me what's available

```
GOAL=GROUND_SERVICE
TRANSPORT_TYPE=rental car
CITY_NAME=atlanta
```

what does co mean

```
GOAL=ABBREVIATION
AIRLINE_CODE=co
MEANING=mean
```

Figure 3.2: Examples of frame-based labelling in ATIS-3. Each example has a goal which represents the task which the utterance relates to. In these examples we present the three main topics in ATIS 3: asking for a flight, asking for a rental car or asking about the task itself.

[He and Young, 2006; Zettlemoyer and Collins, 2007]. In this work we explore the case when such resources are not available.

In the training part of the ATIS corpus there are 21 *goals* types. One set of these types is related to asking for information about flights, aircraft or airlines as in the first example utterance. Another set relates to asking information about renting a car, as in the second example utterance. The third set is relates to questions about the task, as in the third example utterance (see section A.1). The training corpus has 95 different types of slots. On average there are 3.39 slots per utterance with a median of 4 (see Figure A.2). In total the corpus has 15,171 instances of slots, of which more than the half correspond to only 2 types (i.e., `FROMLOC.CITY_NAME` and `TOLOC.CITY_NAME` which denote origin and destination cities). Of the 95 types, there are only 19 that

occur more than 100 times. This situation makes the slot distribution unbalanced. Most of the slots occur only rarely, but there is a core of slots which are very common (see Figure A.3). The goal distribution has a similar property (see Figure A.4).

3.2 Communicator Corpus

The Communicator corpus [Bennett and Rudnicky, 2002] is a collection of spoken dialogues between a human user and different dialogue systems in the domain of flight planning. Some examples of the utterances are presented in Figure 3.3.

1. *i wanna go from denver to indianapolis on november eighteenth*
2. *i don't wanna hotel in chicago*
3. *i don't want to go to chicago i want to go to st louis*

Figure 3.3: Example of utterances from Communicator corpus [Bennett and Rudnicky, 2002].

In a similar way to in ATIS, Communicator is divided into parts. Originally, it was divided into two parts: training and testing. Since we needed a development corpus for training we further divided the training part into new training and development parts. Table 3.2 presents the sizes of the different parts of the corpus.

Corpus	Number of utterances
Training	11,502
Development	1,200
Testing	1,177

Table 3.2: Number of utterances in parts of the Communicator corpus.

3.2.1 Semantic representation

This version of the Communicator corpus was developed by He and Young [2006]. It is similar to the ATIS corpus in that it is composed of transcriptions of the utterances and their frame-based semantic representations. This version of the corpus was hand labelled. In this case, the corpus is labelled with *goals* and *slot-values* (see section 1.2).

Figure 3.4 presents the frame-based labelling for the previous examples of utterances. Each one of the utterances encodes a goal and a set of slot-values. The slots can be composed of one or two semantic concepts, as presented in the ATIS corpus (see previous section). In the second and third utterances of our examples we notice how negation is handled: there is a special slot, `NEGATIVE`, to indicate negation. However, there is a problem with this representation. In the second example the right interpretation involves negating all slots which follow the negation slot. But in the third example the right interpretation is to negate only the first slot which follows the negation. As stated in the first chapter, the lack of scope in the slot-value semantic representation is a known issue. In Chapter 7 we explore a more complex semantic representation which has an extended notion of scope.

In the training part of the Communicator corpus there are 20 *goals* types. One set of these types relates to asking for information about flights. Another set relates to asking for information about renting a car or booking into a hotel. The third set relates to questions about the task. The training corpus has 108 types of slots. On average there are 3.06 slots per utterance with a median of 2 (see Figure A.6). Similar to ATIS corpus, this is an unbalanced corpus. It has $26,117$ instances of slots, but 7 of the slots types happen more than 100 times and these account for 68% of the corpus (see Figure A.7). When compared with ATIS corpus, although the Communicator corpus is bigger than ATIS in terms of the number of pairs of utterances and semantic representations, the language is less complex, since utterances are shorter and contain fewer slots.

3.3 TownInfo

The TownInfo corpus [Georgila et al., 2005a] is a collection of spoken dialogues between a human user and a dialogue system in the domain of tourist information. The user asked for information about services such as restaurants, hotels and/or bars. Some examples of utterances in the TownInfo corpus are presented in Figure 3.5.

For experimental purposes the TownInfo corpus was split into three parts: training, development and testing. Table 3.3 presents the sizes of the different parts of the corpus. On average the length of the utterances in the training part is of 3.60 words, with a median of 6 words. This is a small corpus, mainly because it was collected during the TownInfo system development process. For the case of TownInfo we paid special attention to making sure that the splits were based on users rather than on

i wanna go from denver to indianapolis on november eighteenth

```
GOAL = AIR
FROMLOC.CITY_NAME = denver
TOLOC.CITY_NAME = indianapolis
MONTH_NAME = november
DAY_NUMBER = eighteenth
```

i don't wanna hotel in chicago

```
GOAL = HOTEL
NEGATIVE = don't
HOTEL = hotel
CITY_NAME = chicago
```

i don't want to go to chicago i want to go to st louis

```
GOAL = AIR
NEGATIVE = don't
TOLOC.CITY_NAME = chicago
TOLOC.CITY_NAME = st louis
```

Figure 3.4: Frame-based labelling examples from Communicator corpus. Each example has a goal and a set of slot-values. In particular, these examples show the way negation is represented within the frame-based representation.

utterances (as it is the case in ATIS and Communicator corpora). This made splits more realistic given than the SLU module will be used by new users. This setup provided a stricter evaluation than ATIS and Communicator corpora because if the SLU is not able to identify a semantic concept, a common reaction from the user when a mistake occurs is to keep rephrasing their utterances using the same concept. This situation increases the number of mistakes that the system makes since in many cases it cannot correctly identify this particular concept. The setup of the TownInfo corpus provides a realistic scenario in terms of developing a SLU module for a dialogue system. With the TownInfo corpus, we tested approaches to a scenario in which the dialogue system developer has to create a SLU module with minimal resources, a small amount of data

1. *looking for a french restaurant in the centre of the town*
2. *could you repeat the phone number please*
3. *garden*

Figure 3.5: Example of utterances from TownInfo corpus [Georgila et al., 2005a].

and with a tailored semantic representation.

Corpus	Number of users	Number of utterances
Training	12	1,710
Development	3	445
Testing	4	730

Table 3.3: Number of utterances and users in the parts of the TownInfo corpus.

3.3.1 Semantic representation

The TownInfo corpus is composed of pairs of transcriptions of utterances and their semantic representations. Additionally, the TownInfo provides automatic transcriptions of the utterance. Each semantic representations are specifically tailored to the TownInfo system. It consists of a triplet of speech act, slot and value. Figure 3.6 presents the triplet labellings for the examples of utterances in Figure 3.5. For instance the first utterance of our examples has three triplets. The first triplet consists of the speech act `provide_info`, the slot `food_type` and the value `french`. In the case of the TownInfo corpus, the task/goal is encoded as a triplet (see the second triplet of the example). There are three factors which make this semantic representation different from those used in the ATIS and Communicator corpora:

1. Each slot-value pair has a speech act related to it. For instance, the first utterance of our examples has three triplets, of which each first element is a speech act.

2. The value is not necessarily a sequence of words from the utterance. For instance, in the second utterance of our examples, the value is `null`, which is not present in the utterance.

3. The corpus contains instances of utterances without meaning. For instance, the third utterance of our examples did not have any meaning in the dialogue.

looking for a French restaurant in the centre of the town

`(prov_info,food_type,french),`

`(prov_info,task,restaurant),`

`(prov_info,region,centre)`

could you repeat the phone number please

`(wh_question,phone_number,null)`

garden

`( )`

Figure 3.6: Example of semantic triplet labellings from the TownInfo corpus. The first utterance shows a set of tripleits, with each one having a word of the utterance as its value. The second utterance shows a triplet whose value is not part of the utterance (i.e., `null`). The third utterance shows an empty triplet: this means there is no valid semantic representation for this utterance.

In the training part of the TownInfo corpus there are 1.30 triplets on average, with a median of 1 (see section A.3). In the corpus there are $1,933$ instances of triplets, comprised of 63 types. In these triplets, the corpus labels only 4 types of speech acts. There are 17 types of slots, of which 16 are filled with values of sequences of words in the utterance and 13 not. These slots take 48 types of values, but there are only 18 which are not present as sequences of words in the utterance.

3.4 PropBank and NomBank

The PropBank [Palmer et al., 2005] and NomBank [Meyers et al., 2004] corpora label the Semantic Roles of predicates of the Penn TreeBank [Marcus et al., 1993]. Propbank labels the verbal predicates and NomBanks labels the nominals (associated with nouns). Semantic roles denote the relation that a predicate has to its syntactic frame (i.e., the set of prototypical syntactic constituents associated with a verb or noun). For instance, in the sentence *Ms. Haag plays Elianti* the predicate `play` has a syntactic frame with two arguments, the noun phrases: *Ms. Haag* and *Elianti*. The first noun phrase defines the semantic role as *actor* while the second one defines the semantic role of the *character played*, in this case the Elianti. The set of semantic roles which defines a distinct use of a predicate is called the *roleset*. The roleset associated with its

syntactic frame is called the *Frameset*. PropBank and NomBank collect the Framesets of the predicates which appear in the Penn TreeBank.

Both corpora list the Framesets for the Penn TreeBank and extend them with examples showing the different realisations of the syntactic frames. Figure 3.7 shows two examples of Framesets. The first one corresponds to the verb *play*. In this case, we show the Frameset for playing a role which is the meaning of the verb used in the example *Ms. Haag plays Elianti*. In this case, the Frameset is identified by the label `play.02` which is the lemma plus an identifier. The identifier is the number 02, and this means that this verb has at least two meanings. The other meaning, identified as `play.01`, is for playing a game. The Frameset for `play.02` shows two roles: *Arg*0 and *Arg*1. The first one corresponds to the agent of the action, and the second one the role played. Although the *actor* and *role* labels appear in the corpus, these only have the goal of documenting the Frameset and they does not correspond to any theoretical framework. The example also includes a sentence example.

The second example in Figure 3.7 is *hand.01*, which means something participating or controlling. This predicate differs from the first example in that it is assigned to a noun rather than a verb. This is the main difference between PropBank and NomBank. PropBank specialises in verbal predicates while NomBank focuses on nominal predicates. In this example, the Frameset *hand.01* has two arguments. The first role corresponds to the agent, and the second role to the theme, activity or entity controlled. The Frameset example also includes a sentence example.

PropBank and NomBank uses the labels from `Arg0` to `Arg5` to label the main arguments of a predicate. In Addition to the arguments of verb, the PropBank and NomBank corpora labels the adjunct-like arguments, to signal characteristics as follows: location, extent, discourse connectives, general-purpose, cause, time, purpose, manner and direction. It also labels the negation and model verbs as part of the adjunct-like arguments, although they are not actually adjunct-like phenomena.

3.4.1 CoNLL-08 corpus

In this work, we used the corpus developed for the CoNLL-08 Shared task for Joint Parsing of Syntactic and Semantic Dependencies [Surdeanu et al., 2008]. This corpus joined both the PropBank and NomBank annotations of the Penn TreeBank and extended them with extra annotations. The purpose of the corpus was to jointly perform Dependency Parsing and Semantic Role Labelling. In our case, though, we only used

Frameset: **play.02**
"play a role"
Arg0: actor
Arg1: role
Example: [$_{Arg1}$*Abbie Hoffman*], *in this case, is played by* [$_{Arg0}$ *Paul Lieber*].

Frameset: **hand.01**
"participate/control/possess/lose-control/possession/credit"
Arg0: agent, entity participating
Arg1: theme/activity/controlled
Example: [$_{Arg1}$ the enterprises] still in [$_{Arg0}$ state] hands

Figure 3.7: Examples of Frameset. The first example was extracted from PropBank. It shows the propositional predicate `play.02`. The second example was extracted from NomBank and labels the nominal predicate `hand.01`.

it for Semantic Role Labelling.

The corpus consists of Penn Treebank sentences aligned with their Semantic Role Labels. The corpus identifies which words are predicates, and which are arguments. Additionally, it links such arguments with their role. This version of the corpus is different from the previous version [Carreras and Márquez, 2005] in that CONLL-08 uses Dependency parses [Buchholz and Marsi, 2006] to signal the syntactic argument of the sentence rather than phrase-based constituents.

Figure 3.8 presents an example of SRL labelling for the CoNLL-08 Shared task for the utterance *Ms. Haag plays Elianti.*. The columns contain the following information:

1. Position of the word in the utterance (e.g., 3).
2. Word (e.g., plays).
3. Lemma of the word (e.g., play).
4. Gold POS tag of the word (e.g., VBZ).
5. State-of-the-art POS tag (e.g., VBZ).
6. Part of the word if it is a composed word using hyphens (e.g., plays).
7. Lemma of part of the word if it is acomposed word using hyphens (e.g., play).

8. POS tag for segment of the word in case if it is a composed word using hyphens (e.g., VBZ).

9. Head dependency (e.g., 0).

10. Dependency relation (e.g., ROOT).

11. Roleset (e.g., play.02).

12. Role for the words which are arguments. Each column belows describes the arguments for each predicate in the order they appear in the sentence.

The corpora is extended with part of speech (POS) tags and lemmas (LEMMA), both of them obtained with state-of-the-art POS taggers and lemmatisers [Gimenez and Márquez, 2004; Fellbaum, 1998].

```
1   Ms.     ms.     NNP NNP Ms.     ms.     NNP     2   TITLE   _       _
2   Haag    haag    NNP NNP Haag    haag    NNP     3   SBJ     _       A0
3   plays   play    VBZ VBZ plays   play    VBZ     0   ROOT    play.02 _
4   Elianti elianti NNP NNP Elianti elianti NNP     3   OBJ     _       A1
5   .       .       .   .   .       .       .       3   P       _       _
```

Figure 3.8: Example of the sentence *Ms. Haag plays Elianti* from the CoNLL-08 corpus.

The example presents the predicate `play.02(Ms.Haag,Elianti)`. The Roleset `play.02` is assigned to the verb *plays* in position 3. Since there is only one predicate, there is only one column following the predicates column (number 12). This column signals two roles: $A0$ and $A1$. The first role is assigned to the dependency tree the position 2 and the second role to the word in tree 4. In the first role the dependency tree expands to the phrase *Ms. Haag* and for the second role the dependency tree expands to the phrase *Elianti*.

3.5 Summary

We have presented the five corpora we used in this work: ATIS [Dahl and et al., 1994], Comunicator [Bennett and Rudnicky, 2002], TownInfo [Georgila et al., 2005a], PropBank [Palmer et al., 2005] and NomBank [Meyers et al., 2004]. For the latter two,

we used the version provided in CoNLL-08 [Surdeanu et al., 2008]. Each of these corpora has properties which allows us to investigate the adequacy of Markov Logic for semantic processing. The first three corpora are relatively small and were collected in the context of dialogue systems. They consists of utterances annotated with shallow semantic representations. However, none of these corpora incorporate the notion of scope. On the other hand, the PropBank and NomBank corpora are relatively bigger corpora and they provide more complex semantic representations for the Semantic Role Labelling (SRL) task.

Corpus	ATIS-3	Communicator	TownInfo	CoNLL-08
Relative size	Small	Small	Small	Large
Type of language	Spoken	Spoken	Spoken	Written
Domain	Flight booking	Flight planning	Tourist information	Wall Street Journal
Semantic representation	Frame-based	Frame-based	Triplets	Semantic roles
Extra information	Lexical classes	Lexical classes	None	POS tags, lemmas and dependency trees

Table 3.4: Summary of properties of corpora used in this thesis.

We will use the corpora in the following way. The ATIS and Communicator corpora will be used to create a model which is directly comparable with state-of-the-art approaches (see Chapter 5). Additionally, we will present further modifications to this model (see Chapter 6). We will use the PropBank and NomBank corpora to perform Semantic Role Labelling, to test the adequacy of Markov Logic for handling more complex semantic representations than the frame-based representations of previous corpora (see Chapter 7). Finally, we will use the TownInfo corpus to test the development of a Spoken Language Understanding (SLU) module using Markov Logic (see Chapter 8). In this case, the developer of the module has access to limited resources and only a small amount of labelled data. Additionally, the semantic representations of this SLU module are tailored to a specific dialogue system.

Chapter 4

Statistical models for SLU

This chapter presents the theoretical framework for the two main approaches we use in this thesis: Hidden Vector State [HVS, He and Young, 2005] and Markov Logic [ML, Richardson and Domingos, 2006]. In particular, in this chapter we focus on the Spoken Language Understanding (SLU) task which is a closed domain task. Firstly, we start by introducing the HVS framework which is an extension of Hidden Markov Models (HMM). In particular, the HVS model was proposed with the purpose to be used in the SLU task. Second, we present the ML framework which combines First Order Logic with Markov Networks. The ML framework was proposed as a means to handle uncertainty in First Order Logic, however recent developments of the framework make it possible to be used for Natural Language Processing tasks [Riedel, 2008; Riedel and Meza-Ruiz, 2008]. In our case, we model the SLU task within the ML framework.

The HVS model and the ML model have a different approach to the statistical modelling. HVS is a generative model. This means that it models the outcome of a stochastic process given a set of observations as a joint probability (i.e., $P(Y,X)$ where Y is the outcome and X the observations). On the other hand, ML is a discriminative model which models such process as a conditional probability (i.e., $P(Y|x)$). This difference contributes to some of the main pros and contras of the frameworks. The HVS statistical elements are probabilities which makes it easy to understand and on which we can use some of the techniques to model adaptation when faced with a new domain. However, HVS models are hard to extend with new elements. In contrast, ML can be easily extended with features because these are at the core of the discriminative models, but since the features use weights as main paradigm to represent the stochastic process the intuition behind the probabilities is lost and model adaptation is harder.

The outline of this chapter is as follows. In section 4.1 we introduce the HVS

model. In particular, we present the formal definition of the HVS model and the parameter estimation used with it. Section 4.2 introduces the theory behind the ML approach. In particular, in this section we present the ML algorithms used for inference and learning and a graphical representation for the Markov Networks generated by the ML framework. The definitions presented in this chapter are used in chapter 5, where we present the main results of a first set of experimentation using both frameworks. The goal of those experiments is to compare the performance of ML with an acceptable baseline, in this case using the HVS framework.

4.1 Hidden Vector State

The Hidden Vector State model is an extension of Hidden Markov Models (HMMs). They are different to HMMs in two ways. Firstly, an HVS model consists of vector states rather than the single states in an HMM. This vector is equivalent to a snapshot of a stack in a push-down automaton. Second, the transition probability from vector to vector is factorised into two probabilities:

- The probability of popping n states of the vector/stack.
- The probability of pushing one state into the remaining vector/stack.

From now on, I will be using the terms *vector state* and *stack* interchangeably. Figure 4.1 shows a graphical model representing the transition from one stack of states $\mathbf{C_{t-1}}$ to the next one $\mathbf{C_t}$. As can be seen, each stack of states emits an observation (o_T), in a similar manner to an HMM state emits an observation. This means that the observation in an HVS is conditioned by the vector state stack. In the figure, it can be noted that the stacks share states between them (e.g., $C_n^t = C_m^{t-1}$). These are the states which were not popped from the first stack. In addition to these elements, the second stack contains an element which was pushed into it. While worst case complexity is *NP*-hard, tractability is controlled by limiting the size of the stacks.

The HVS framework was proposed with the frame-based semantic representation in mind. The speech act and the semantic concepts are the stacks and the words of the utterance are the observations. However, with this correspondence, not all the words of the utterances have an assigned stack: the words which are not values are missing a stack configuration. He and Young [2005] propose that this configuration is unknown but it is related to a semantic trees on which the nodes are semantic concepts and the leaves are words. Figure 4.2 illustrates one of the possible semantic tree corresponding

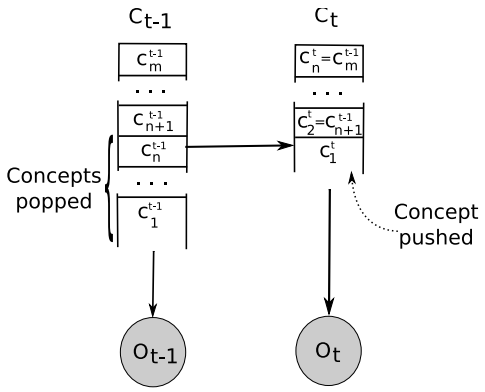


Figure 4.1: Transition between two stacks of states in the HVS model. During the transition n concepts are popped from the origin stack and 1 concept is push. The rest of the stack stays the same.

to our example 1.1. This semantic tree generates the sequence of stacks presented in 4.3. Notice that each transition between stacks follows the definition of the HVS: it pops n semantic concepts (i.e., states) and pushes one. The goal of the HVS model when applied to the frame-based semantic representation is to model the sequence of the stacks of the semantic tree. Within these stacks there would be stacks which correspond to slots of the semantic representation. We will see how to address the issue of the unknown semantic tree in section 5.1.

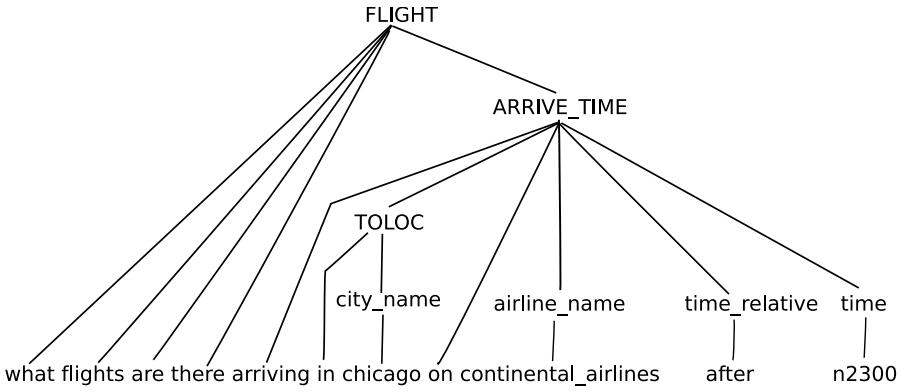


Figure 4.2: Semantic tree for the utterance *what flights are there arriving in chicago on continental_airlines after 2300*. Terminal nodes are in lowercase and non-terminal in capitals.

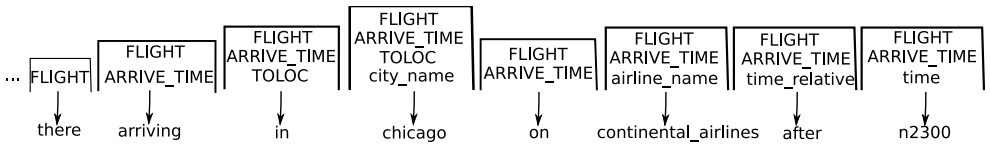


Figure 4.3: Sequence of stacks representing a semantic tree for utterance example.

The HVS model is generative model which decomposes the joint probability of a sequence of stacks $\mathbf{C}$ and a sequence of observations $\mathbf{O}$ as follows:

$$P(\mathbf{C},\mathbf{O}) = \prod_{t=1}^{T} P(n_t|\mathbf{C}_1^{t-1},\mathbf{O}_1^{t-1})P(c_t[1]|\mathbf{C}_1^{t-1},\mathbf{O}_1^{t-1},n_t)P(o_t|\mathbf{C}_1^{t},\mathbf{O}_1^{t-1}) \tag{4.1}$$

where:

- $\mathbf{C}_1^t$ denotes a sequence of stacks ranging from the first stack to the stack in the position t.
- $\mathbf{O}_1^t$ denotes a sequence of observations ranging from the first observation to the observation in the position t.
- n_t denotes the number of elements popped to arrive to the position t.
- $c_t[1]$ is the salient element of the stack at the position t.
- o_t is the observation at the position t.

Equation 4.1 does not have any independence assumption between stacks of states. The first probability of the product corresponds to the probability of popping n elements given the sequence of stacks and observations from the start to the time $t-1$. The second probability corresponds to the probability of pushing a state $c[1]$ given the whole sequence of stacks, observations from the start to time $t-1$, and the number of popped elements. Finally, the last probability of the product corresponds to the probability of emission of the observation o given the sequence of stacks from the start to time t and the observation from the start to time $t-1$. However to generate a model for the Equation 4.1 is impractical since having the probabilities over all previous elements of the sequence would result in a sparse distributions. For this reason He and Young [2005] propose to assume the first order Markov assumption: namely, a current stack depends only on the previous stack. Additionally their model approximates the

elements of Equation 4.1 into the following equations:

$$\begin{aligned} P(n_t|\mathbf{C}_1^{t-1},\mathbf{O}_1^{t-1}) &\approx P(n_t|c_{t-1}) \quad (pop) \\ P(c_t[1]|\mathbf{C}_1^{t-1},\mathbf{O}_1^{t-1},n_t) &\approx P(c_t[1]|c_t[2\cdots]) \quad (push) \\ P(o_t|\mathbf{C}_1^{t},\mathbf{O}_1^{t-1}) &\approx P(o_t|c_t) \quad (emission) \end{aligned} \tag{4.2}$$

The formulae 4.2 are the expressions for the pop and push operation and the emission of an observation, respectively. These formulae assume that the number of elements to pop depends only on the previous state (i.e., $n_t|c_{t-1}$), that the element to push depends on the elements which were not popped from the previous stack (i.e., $c_t[1]|c_t[2\cdots]$), and that the observations only depends on the current stack (i.e., $o_t|c_t$). In the case of the observations, this is the same assumption done by the HMM model. For the case of the popping and pushing operations, the intuition is that the configuration of the stack at time t depends on the pop and push operations performed on the stack at time $t-1$. Since the formulae 4.2 simplifies the conditions of formula 4.1 the size of the original distributions for Equation 4.1 has been reduced and is now practical to implement because the distributions only depend on elements of the previous stack, rather than the whole sequence. However, we lost generalization when there are dependencies between two stacks which are separated by more than one position. For this case, we expect that the state configuration helps to track these dependencies.

4.1.1 Parameter estimation

Equation 4.2 defines the main components of the HVS model: the probability distributions of the pop, the push operations and the emission of an observation. Those distributions have to be collected from examples. In order to estimate these probabilities He and Young [2005] suggest the following formulae for a given λ HVS model and example conformed by sequence of observations $\mathbf{O}$:

$$\begin{aligned} P(n|c') &= \frac{\sum_t P(n_t = n, c_{t-1} = c'|\mathbf{C},\lambda)}{(\sum_t P(c_{t-1} = c'|\mathbf{C},\lambda))} \\ P(c[1]|c[2\cdots]) &= \frac{\sum_t P(c_t = c|\mathbf{C},\lambda)}{(\sum_t P(c_t[2\cdots] = c[2\cdots]|\mathbf{C},\lambda))} \\ P(o|c) &= \frac{\sum_t P(c_t = c|\mathbf{C},\lambda)\delta(o_t = o)}{(\sum_t P(c_t = c|\mathbf{C},\lambda))} \end{aligned} \tag{4.3}$$

The intuition behind these formulae is that given a current model λ, we can estimate the probability of the distributions. We do this by counting the number of times the outcome and the condition we are interested in happened (i.e., the numerator in the three

expressions) divided by all the times our conditional happened for a given sequence (i.e, the denominator in our three expressions). For instance, given model λ the probability distribution for the pop operation $P(n|c')$ is calculated by counting how many times the model transitioned between stacks popping n states from the stack c'. This count is then divided by count of how many times the model passes by the stack c'. The new probability we obtain forms part of a new model λ' which can be used again to calculate the distributions. In this iterative fashion we can calculate the distributions until the model λ' converge. This procedure is defined in the *forward-backward* algorithm [Rabiner, 1989]. In practical terms, the estimation of the HVS model is called *training*.

Estimating the model is half of the work. Once we have a model, we want to be able to predict the sequence of stacks for an sequence of observations **O**. The Viterbi versions of the *forward* or *backward* algorithms can be used to identify the sequence of vector states which maximize the joint probability defined in Equation 4.2 He and Young [2005]. In practical terms the calculation of the sequence which maximizes the probability of the sequence of observation is called *labelling*.

4.2 Markov Logic

In this section, we present the SLU task as a case of structured classification in the context of discriminative methods. In a discriminative method the outcome of a problem depends on some observations. In terms of the SLU task the outcome is the semantic representation, for instance slots, and the words of a utterances are considered observations. When besides the dependencies of the outcome and observations there are parts of the outcome that depend on other parts of the outcome we say the problem is a structured classification [Sutton and McCallum, 2006]. Central to the definition of HVS is the first order Markov assumption which states that the current slot depends only on the previous one. This situation points out we can define SLU task as a case of structured classification.

We call the dependencies between outcome and observation *local*, and the dependencies between parts of the outcome *global*. Nowadays to create discriminative models that exploit local dependencies is relatively straight forward (for instance we could create a Maximum Entropy model using off-the-shelf software). However, to model global dependencies is still an active field of research. In this work, we focus on the ML framework to model the SLU task as a structured classification. ML is a First Or-

der Logic (FOL) probabilistic language which is used to instantiate Markov Networks (MN) of repetitive structure. The intuitions about the structure of the semantic representation and the dependencies among the elements of this structure are represented using FOL formulae. These formulae are translated into a MN which is used to learn the task. Since the formulae control the creation of a MN it is relatively easy to test new dependencies in a model: this only requires the addition of new formulae.

Markov Logic [ML, Richardson and Domingos, 2006] is a Statistical Relational Learning (SRL) language based on First Order Logic (FOL) and Markov Networks (MN). It can be seen as a formalism that extends FOL to allow formulae that can be violated with some penalty. Or from an alternative point of view, as an expressive template language that uses FOL formulae to instantiate Markov Networks of repetitive structure.

The core of ML is the Markov Logic Network (MLN). An MLN M is the knowledge base composed by the pairs $\{(\phi_i, w_i)\}$ where each ϕ_i is a formula in FOL and w_i is a real number. The MLN has a similar function to a knowledge base in FOL which describes the set of possible words that for which all its formulae are satisfied. However, the MLN knowledge base instead of classifying possible words as either consistent (all formulas are satisfied) or inconsistent (some are not) it maps each possible word to a probability. This allows us to model uncertainty in our beliefs about the world.

In order to define a formula ϕ we need a vocabulary. A vocabulary consists of the objects of the domain, constants, and the relations over the objects, predicates. For example, for the SLU task, the constants could be the orthography of a word (e.g., *chicago*, *on*, *after*), or its position (e.g., 1, 2, 3), and a predicate would be *word/2* which represents a relation between a position and the orthography. The 2 in the predicate represents the arity of the predicate. For this example, the first argument is the position and the second one the orthography. Using this predicate we can define a possible world which represents the utterance *what flight are there arriving in chicago on continental airlines after n2300*:

$$\{word(1,what), word(2,flights), word(3,are), word(4,there),$$
$$word(5,arriving), word(6,in), word(7,chicago), word(8,on),$$
$$word(9,continental\_airlines), word(10,after), word(11,2300)\}$$

With the vocabulary, we can specify FOL formulae for the MLN. We will use open formulae and assume each free variable to be universally quantified. These formulae describe the structural relations of a particular problem. For instance, in the case of

SLU the formula:

$$\phi_1 : word(p_1, w_1) \wedge slot(p_1, s_1)$$

defines a relation between a word w_1 of an utterance at the position p_1 with a slot s_1 in the same position. This relation is among all possible pairs of words and slots in a given position: we will see below that this relation would be weighted depending on the pair word-slot. This weight encodes the preference of a certain pair over others. This relation does not impose any structure to the model. In sections 6.2.1.2 and 7.2.2 we will see examples of how to use the FOL of ML to constrain the structure of the semantic representation.

Next we introduce some definitions. We denote the number of free variables of a formula ϕ with n_ϕ thus for our example n_{ϕ_1} is 3 . A formula that contains a single predicate and nothing else is called an atom. As defined for FOL, we say that a possible world W satisfies a formula ϕ and write $\models_W \phi$ if and only if ϕ is true in W. With this in mind, we say

Definition 1 *A* grounding $\phi[v_1/c_1, \cdots, v_{n_\phi}/c_{n_\phi}]$ *represents the substitution of each variable* v_i *with the constant* c_i*, which we abbreviate as* $\phi[\mathbf{v}/\mathbf{c}]$.

A formula that does not contain any variables is said to be grounded. For instance, a grounding of the formula ϕ_1 is

$$\phi_2 : word(1, chicago) \wedge slot(1, fromloc.city\_name)$$

Notice that a possible world is a set of ground atoms. For instance, a possible word for ϕ_1 is:

$$W_1 : \{word(1, chicago), slot(1, fromloc.city\_name)\}$$

Definition 2 *We define* $\mathcal{Y}_{P,C}$ *as the set of all possible worlds we can construct using a set of predicates P and a set of constants C.*

Table 4.1 presents some of these concepts with examples.

The goal of the MLN is to map a possible world to a probability which measure its plausibility. In an MLN the formulae ϕ define the possible worlds and the weights are

Concept	Example	$n_{example}$
Constants (C)	1,2,3...*chicago*,*what* ...	Not applicable
Predicate (P)	$word/2, slot/2$	Not applicable
Formula (ϕ)	$word(v_1, v_2) \wedge slot(v_1, v_3)$	3
Atom ($\mathcal{Y}$)	$word(v_1, v_2)$	2
Grounding	$[v_1/1, v_2/chicago, v3/fromloc.city\_name]$	Not applicable
Ground formula	$word(1, chicago) \wedge slot(1, fromloc.city\_name)$	0
Ground atom	$word(1, chicago)$	0
Possible worldi (W)	$word(1, chicago), slot(1, fromloc.city\_name)$	Not applicable
$\mathcal{Y}_{P,C}$	$word(1, chicago), slot(1, fromloc.city\_name)$ $word(1, denver), slot(1, fromloc.city\_name), \ldots$ $word(1, chicago), slot(1, toloc.city\_name), \ldots$	Not applicable
MLN ($M = (\phi, W)$)	$(word(v_1, v_2) \wedge slot(v_1, v_3), w_1)$ $(word(v_1 + 1, v_2) \wedge slot(v_1, v_3), w_2)$ ...	

Table 4.1: ML definition with notation and examples

used to map them as a probability. For this purpose, an MLN M defines a log-linear probability distribution over possible worlds $\mathbf{y} \in \mathcal{Y}_{P,C}$ as follows:

$$p(\mathbf{y}) = \frac{1}{Z} exp\left(\sum_{(\phi,w)\in M} w \sum_{c\in C^{n_\phi}} \mathbf{f}_c^\phi(\mathbf{y})\right) \tag{4.4}$$

where the feature function $\mathbf{f}_c^\phi(\mathbf{y})$ returns 1.0 when the grounding $\phi[\mathbf{v}/\mathbf{c}]$ satisfies the formula ϕ, otherwise it returns 0.0. $\mathbf{C}^{n_\phi}$ is the set of all tuples of constants we can replace the free variables in ϕ with. Z is a normalisation constant.

The definition in Equation 4.4 joins FOL with the stochastic part of the MLN. The feature function $\mathbf{f}_c^\phi$ translates subsets of constants c of the possible world $\mathbf{y}$ into 1s and 0s. For instance, for the formula ϕ_1 the feature function $\mathbf{f}^{\phi_1}_{(chicago,fromloc.city\_name)}$ returns 1.0, but $f^{\phi_1}_{(car,fromloc.city\_name)}$ returns 0.0, since *car* is not a city and the formula is not satisfied. These 1s are counted by the second sum of the equation. This count is then multiplied by its corresponding weight w. In this way, each formula in the MLN is weighted in terms of its features. When all these weights are summed we obtain a score for the possible world $\mathbf{y}$. In order to translate this score into a probability, we divide by the normalization factor Z. However, this normalization implies obtaining the scores for all possible worlds, which makes it intractable [Sutton and McCallum, 2006]. However, in the next subsection we see how to avoid this problem.

The intuition behind Equation 4.4 is that possible worlds which satisfy all or most of the formulae ϕ would be mapped to a higher probability. This is because of two factors: the amount of features each formula has and the weights W. The amount of features depends on the instance of the problem we are modeling. For instance, in the SLU task, the amount of features depends on the current utterance we process. However, we can tune the weights to match a distribution of possible worlds which is consistent with the real world. This is the learning problem. Once we have a set of weights, we can ask for the possible world that maximises the probability for Equation 4.4 given that we partially observe the possible world. This is the inference problem. In the context of SLU this corresponds to the labeling problem. In particular, for the learning problem we use 1-best Margin Infused Relaxed Algorithm (MIRA) online algorithm [Crammer and Singer, 2003] and Cutting Plane Inference (CPI) algorithm [Riedel, 2008] for the labelling problem, which in the context of ML is known as *inference*.

4.2.1 Labelling/Inference

The inference problem consists of asking which is the set of ground atoms $\hat{\mathbf{y}} \in \mathcal{Y}_{H,C}$ with maximum a posteriori probably (MAP), this is:

$$\hat{\mathbf{y}} = \arg\max_{\mathcal{Y}_{H,C}} p(\mathbf{y} \mid \mathbf{x}) \tag{4.5}$$

where P, H are two set of predicates that satisfy $P = H \bigcup O$ and $H \bigcap O = \emptyset$, and C are the constants. We call the set O observed predicates and the set H hidden predicates. In practical terms, the hidden predicates are the predicates which specify the outcome, while the observed specified the observations.

On the other hand the conditional version of Equation 4.4 is defined as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(x)} exp\left(\sum_{(\phi,w)\in M} w \sum_{c\in C^{n_\phi}} \mathbf{f}_c^\phi(\mathbf{y},\mathbf{x})\right) \tag{4.6}$$

The only modification is on the feature function, which distinguishes between hidden and observable predicates. With this version we can reformulate Equation 4.5. However, notice that finding the set of hidden ground atoms that maximize the conditional probability is equivalent to finding the hidden ground atoms with the maximum weight, this means we can ignore the normalization factor Z and the exponential in Equation 4.6. With this in mind we rewrite Equation 4.5 as:

$$\hat{\mathbf{y}} = \arg\max_{\mathcal{Y}_{H,C}} s(\mathbf{y},\mathbf{x})$$

where:

$$s(\mathbf{y},\mathbf{x}) = \sum_{(\phi,w)\in M} w \sum_{c\in C^{n_\phi}} \mathbf{f}_c^\phi(\mathbf{y} \mid \mathbf{x}) \tag{4.7}$$

Alternatively, Equation 4.7 can be interpreted as a discriminant or scoring function. This means that given the two sets of ground atoms, the hidden and the observed, it measures how well they couple given an MLN M. This interpretation plays a relevant role for the learning and inference algorithms.

We define two types of formulae: local (when a formula ϕ contains only one hidden predicate p), global (when a formula ϕ contains two or more hidden predicates p). The local formulae capture the local dependencies we talked about in the introduction while the global formula capture the global dependencies. But, when the model contains at least one global formula then it corresponds to a structured classification. For instance,, the formula:

$$\phi_1 : word(p_1,w_1) \wedge slot(p_1,s_1)$$

is local since it only contains one hidden predicate *slot/2*, and the formula (which relates two consecutive slots):

$$\phi_1 : slot(p, s_1) \wedge slot(p+1, s_2)$$

is global since there are two hidden predicates *slot/2*.

To find the set of ground atoms when an MLN only contains local formulae is trivial. It corresponds to a classification problem. In this case, we only have to iterate each hidden predicate and get the groundings that maximise the score of the formulae that those predicates appear in. For instance, suppose the MLN described by the three first formulae in the synthetic example described in Table 4.2. This example consists of only local formulae and three hidden predicates p_1, p_2 and p_3 (there are not observed predicates to simplify the example). We need to figure out the groundings of the constants, which can be either a or b, that maximize each of these formulae. In this case, each formula has one feature to facilitate the calculations, and the last column of the table contains the maximum grounding for each formula. In this case, the possible world that maximises this MLN is:

$$\{p_1(a), p_2(a), p_3(b)\}$$

This is the union of the groundings which maximise each local formulae.

The previous case is not of great interest since it does not correspond to a structural classification, because it does not include global formulae. However, to find the maximum score when the MLN contains global formulae is not trivial. In those cases, in order to chose one ground atom is not enough to maximise the formulae in which the predicates explicitly appear, but it has to maximize most of the formulae which the ground atoms relate to. For example, suppose we have the MLN composed by all the formulae in Table 4.2, this is the three local formulae of our previous example plus two new global formulae. If we calculate the ground atoms which maximize each of the ground formula we obtain the sets of the last column of the table. However, we notice that the groundings are not consistent, formula 1 suggest $p_2(a)$, while formula 4 $p_2(b)$. We need to combine the scores for each possible combination. These calculations are shown in Table 4.3.

For this example, the possible world that maximises the score for this MLN is:

$$\{p_1(a), p_2(b), p_3(b)\}$$

with a score of 6. As you can see, this result maximizes the formulae 1, 4 and 5, however it is not the case for formulae 2 and 3.

	ϕ	w	$f_{\mathbf{c}}^{\phi}$	$\hat{y}$
1	$p_1(v_1)$	$w=1$	$f_{v_1/a}$	$p_1(a)$
2	$p_2(v_1)$	$w=1$	$f_{v_1/a}$	$p_2(a)$
3	$p_3(v_1)$	$w=1$	$f_{v_1/b}$	$p_3(b)$
4	$p_1(v_1) \wedge p_3(v_2)$	$w=1$	$f_{v_1/a,v_2/b}$	$p_1(a), p_3(b)$
5	$p_2(v_1) \wedge p_3(v_2)$	$w=3$	$f_{v_1/b,v_2/b}$	$p_2(b), p_3(b)$

Table 4.2: Synthetic MLN with p_1, p_2 and p_3 as hidden predicates and maximum ground atoms per formulae.

$\mathcal{Y}_{H,C}$			Formula scores					Score
$p1$	$p2$	$p3$	1	2	3	4	5	
a	a	a	1	1	0	0	0	2
a	a	b	1	1	1	1	0	4
a	b	a	1	0	0	0	0	1
a	b	b	1	0	1	1	3	6
b	a	a	0	1	0	0	0	1
b	a	b	0	1	1	0	0	2
b	b	a	0	0	0	0	0	0
b	b	b	0	0	1	0	3	4

Table 4.3: Scores for each possible world of MLN in Table 4.2.

4.2.2 Cutting Plane Inference

In terms of efficiency it is not possible to enumerate all possible worlds. When working with real problems with formulae in the order of dozens, predicates in the order of tens, constants in the order of hundreds, and features in the order of thousands, enumeration becomes an inefficient strategy. Riedel [2008] proposes the Cutting Plane Inference (CPI) algorithm to incrementally find the possible world which maximises the score. The intuition behind the algorithm is that it starts with a reduced problem, and little by little increments it until all or most of the formulae are satisfied. The CPI algorithm is shown in Algorithm 1. An important property of this approach is that it is accurate.

The CPI algorithm proceeds in the following manner. As an input, it receives the MLN M, an initial partial grounding $\mathbf{G}^0$ and a set of observed ground atoms x. The initial partial grounding is obtained using a partial MLN consisting of only local formulae. $\mathbf{G}^0$ contains all possible tuples that ground the local formulae. The role of a partial grounding $\mathbf{G}$ is to give an approximation of the first solution. This can be seen in line 4 where the solution y using the previous partial grounding $\mathbf{G}^{i-1}$ is calculated. This solution is compared with the score of our previous best solution y' (line 5). If we found a better solution (i.e., a larger score) we update our best solution y' to y (line 6). For each rule ϕ in M we find the *separation* set (see below) which is added to grounding (line 9). If the current search space $\mathbf{G}^i$ has changed when compared to the previous version (line 11), we go back to line 3, if not the CPI finishes and it returns the previous best solution y'.

An important step of the CPI algorithm is the generation of the separation set. The idea is to detect the global formulae which is not satisfied by the current solution. If the current solution does not satisfies the formula it means there could be a better solution where the formulae are satisfied. In order to try another solution, we increase the amount of tuples in the partial grounding $\mathbf{G}^i$, this is equivalent to add more constrains to the search space. There are two cases to consider in order to add new ground formulae to the partial grounding. If the weight is positive, we add all ground formulae which are not satisfied, therefore next time the CPI algorithm solves the problem, this would contain new constraints which will contribute with more weight. If the weight is negative, we want the contrary effect. We want to consider all ground formulae which are satisfied, so that next time it is solved the weight from the current solution is substracted, and it could find a better solution.

To illustrate the CPI algorithm we go through the algorithm using the MLN of

Table 4.2. As a input we pass the MLN, and the initial grounding $\mathbf{G}^0$ which consist all groundings for the local formulae of the MLN:

$$\mathbf{G}^0 = \{(a)_1, (b)_1, (a)_2, (b)_2, (a)_3, (b)_3\}$$

The sub-indexes in the tuples point out to the predicate they substitute. The set of observed atoms is passed as well, in the example the set is empty. Now, we produce the solution using the grounding $\mathbf{G}^0$ (line 4), since it only contains groundings for local formulae the solution is the same we obtain with the local MLN, this is:

$$\mathbf{y}^0 = \{p_1(a), p_2(a), p_3(b)\}$$

We compare the score of this solution with the previous one, because it is the first one, our solution is better than the previous one which has score 0 and it is assigned as the current best solution y' (line 6). Now, for each formula ϕ in the MLN we find the ground atoms which do not contribute to our solution. You can notice, that formulae 1, 2, 3 and 4 contribute to the weight. But formula 5 is not satisfied therefore we add the possible assignments, this is $(a,a)_5, (a,b)_5, (b,a)_5, (b,b)_5$ to our partial grounding (line 9). Since the partial grounding changed , we need to iterate again (line 11). Then we find the current solution, using the partial grounding G^1 (line 4). In practical terms, we are solving the MLN using the local formulae plus the formulae 5. In this case, the solution is:

$$\mathbf{y}^0 = \{p_1(a), p_2(b), p_3(b)\}$$

This solution comes from a search space similar to the one represented in Table 4.3 but without the column for the formulae 4. In this case, the solution has a partial score of 6. As the score is larger than our previous one of 2 we assign it as our current best solution (line 6). For each formula ϕ of the MLN M we look for the ground predicates which are not satisfied for the current solution. In this case, formulae 2 is not being satisfied, we add it as a grounding to G^1 (line 9). However, this addition does not change G^1 compared with G^2, since these groundings were already included in G^0. Because, G^2 did not changed compared with G^1 we finish the CPI algorithm and we return our current best solution.

$$\mathbf{G}^1 = \{p_1(a), p_2(b), p_2(b)\}$$

As we can see, CPI algorithm did performed more efficient than enumerating all possible groundings. This is done by avoiding to use the groundings for formulae 4.

Algorithm 1 $CPI(M, \mathbf{G}^0, \mathbf{x})$

1: $i \Leftarrow 0$
2: $\mathbf{y}' \Leftarrow \emptyset$
3: **repeat**
4: $\quad i \Leftarrow i+1$ $\mathbf{y} \Leftarrow solve(\mathbf{G}^{i-1}, \mathbf{x})$
5: $\quad$ **if** $s(\mathbf{y}, \mathbf{x}) > s(\mathbf{y}', \mathbf{x})$ **then**
6: $\quad\quad \mathbf{y}' \Leftarrow \mathbf{y}$
7: $\quad$ **end if**
8: $\quad$ **for** $(\phi, w) \in M$ **do**
9: $\quad\quad \mathbf{G}_\phi^i \Leftarrow \mathbf{G}_\phi^{i-1} \bigcup Separate(\phi, w, \mathbf{y}, \mathbf{x})$
10: $\quad$ **end for**
11: **until** $\mathbf{G}_\phi^i = \mathbf{G}_\phi^{i-1}$ $||$ $i >$ maximum iterations

4.2.3 Learning

The learning problem consists of tuning the weights of an MLN M in order to capture the distribution of a set of training examples $(\mathbf{y}, \mathbf{x})$ where y_i and x_i are a set of grounding atoms, y_i are hidden and x_i observed. For learning the weights we use the Margin Infused Relaxed Algorithm [MIRA Crammer and Singer, 2003]) which is shown in 2. This is a general technique for learning weights in a marginal fashion (this is by separating "good" weights from "bad" weights). In this case we want to find the ML weights which lead to right solutions.

Algorithm 2 $MIRA((\phi, w^0), (\mathbf{x}, \mathbf{y}))$

1: $j \Leftarrow 0$
2: **while** $n \leq iters$ **do**
3: $\quad$ **for** $(x_i, y_i) \in (\mathbf{x}, \mathbf{y})$ **do**
4: $\quad\quad \hat{w}^{j+1} \Leftarrow \min ||\, w^{j+1} - w^j \,||$
$\quad\quad$ s.t. $s(x_i, y_i) - s(x_i, y') \leq L(y, y') \forall y_i' \in CPI((\phi, w), \mathbf{G}(x_i), \mathbf{x}_i)$
5: $\quad\quad j \Leftarrow j+1$
6: $\quad$ **end for**
7: **end while**

MIRA is an online, marginal, and conservative learning algorithm. *Online* means it does not need the whole training set in order to create a model. Actually, it creates a model for each instance of the training set, each model based on the previous one (see for loop in line 3). *Marginal* means it looks to create a minimum margin between

good answers and the erroneous ones (see the constraint of optimization in line 4). *Conservative* means it only modifies the weights that during the prediction step had errors (see optimization *min* in line 4). Actually, MIRA is ultraconservative which means it only modifies the weights related to the errors.

MIRA proceeds in the following manner, it receives as input an initial MLN (ϕ, w) together with a training set $(\mathbf{x}, \mathbf{y})$, composed by observations and its outcome. For each element of the training set, it infers the hidden predicates $\mathbf{y}'$ given the observed predicates $\mathbf{x}_i$ and the current state of the MLN. In order to infer this solution it uses the CPI algorithm described in the previous section (last part of the constraint in line 4). The function $\mathbf{G^i}(\mathbf{x})$ creates an initial grounding with the observations required by the CPI algorithm. Line 4 is the update step, it modifies the current weights given the errors of the inferred solution $\mathbf{y}'$. The weights are modified by defining an optimization problem. The objective of the optimization is subject to finding the weights which are closest to the current weights but which separate the right solution y from the parts which are wrong in y' with the margin L. L is an error function which quantitatively tells how good a solution is, for instance, the number of mismatched ground atoms. MIRA needs to iterate n times the training data to find a good model. This is a parameter which is decided during development (line 2).

4.2.4 ML as a graphical model

It is possible to represent Equation 4.6 with a factor graph [Kschischang and Loeliger, 2001]. This representation helps to visualize the relations and models introduced by the MLN. A factor graph for an MLN M is a graph $G = (V, F, E)$ where V are the variable nodes, F the factors and E the edges. Figure 4.4 shows an example of a factor graph. In this case, there are three nodes represented by the circles, one factor represented by the black square, and three edges the lines from factor to node. The factors joins the variables through the edges. We use circles with a darker color for the observed variables, and lighter for hidden variables.

To represent an MLN as a factor graph, a ground atom of the MLN becomes a node in the graph, a ground formula is represented as a factor f, and there is an edge that goes from the factor to each node/predicate which belongs to the formula. The figure 4.5 is the factor graph for the ground formula:

$$\phi_2 : word(1, chicago) \wedge slot(1, fromloc.city\_name)$$

There are two nodes in the graph one for each ground atom. These are connected by

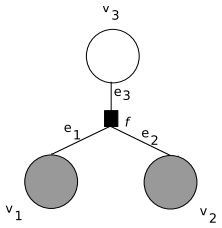


Figure 4.4: Example of a factor graph

the factor ϕ_2 which corresponds to the ground formula. The figure 4.6 presents a factor graph for a formula:

$$\phi_3 : word(0,to) \wedge word(1,chicago) \wedge slot(0,NONE) \wedge slot(1,fromloc.city\_name)$$

which contains more ground atoms. Besides the number of nodes on these two graphs, they differ in their nature. The first factor graph represents a local formula: the hidden node is connected with only observable predicates. The second factor represents a global formula: there are two hidden predicates which connect to the same factor.

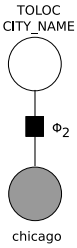


Figure 4.5: Factor graph for ground local formula

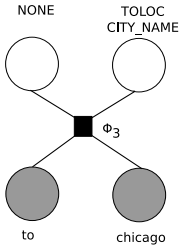


Figure 4.6: Factor graph for ground global formula

As we presented at the start of this section, from an alternative point of view the ML framework can be interpreted as an expressive template language that uses FOL formulae to instantiate Markov Networks of repetitive structure. For instance, the FOL formula

$$\phi_4 : word(p,w) \wedge slot(p,w)$$

specifies that there is Markov Network between the words w and the slots s which share the same position p. In this case, given an utterance the formula generates the graph of repetitive structure presented in Figure 4.7. In this case, this graph represents a local

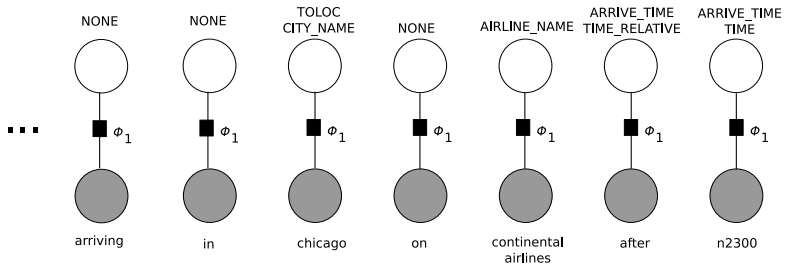


Figure 4.7: Factor graph for a possible world.

formula. There are not factors which connect two hidden nodes. For this reason, each slot can be inferred/labelled independently from the others. In the following chapter, we introduce a Maximum Entropy classifier which follows this strategy. It labels each slot node independently of the other decisions, it only considers the observed nodes.

4.3 Summary

In this chapter we have presented the main theoretical aspect of the two main approaches we use in this thesis. Hidden Vector State (HVS) [HVS, He and Young, 2005] and Markov Logic [ML, Richardson and Domingos, 2006]. For both approaches we have presented the methods to train the models, and to label new data. HVS is a generative model, while ML is a discriminative model. Generative models model a task in term of the joint probability of outcome and observations (see Equation 4.1) while discriminative models proposes a conditional probability of the outcome given the observations (see Equation 4.6). A main difference between HVS and ML is that ML is able to incorporate global dependencies (i.e., more than two hidden variables) as a part of the model. The only global dependency that the HVS model incorporates is the first Markov assumption, which relates a state with the next one.

In chapter 5 we present the considerations we need to take into account in both approaches so that we can perform the SLU tasks with them. Once we present those considerations the chapter presents the results of our implementations. The main goal is to test the performance of the ML for the task of SLU. To do this, we create a baseline using HVS and a local discriminative classifier. These baselines are then compared with a model created using ML.

Chapter 5

SLU with limited resources

This chapter presents our experiments with the Markov Logic [ML, Richardson and Domingos, 2006] approach in order to measure its adequacy to perform Spoken Language Understanding (SLU). In particular, we will focus on a case with limited resources: this is when the system developer only has access to a simple semantically annotated corpus. Our first step will be to create a baseline using the state-of-the-art Hidden Vector State approach [HVS, He and Young, 2005]. This baseline will allow us to compare the ML model with a state-of-the-art result. The second step will be to create a purely local discriminative model of the task which will be our second baseline [MaxEnt, Ratnaparkhi, 1999]. This baseline will help us to compare the ML model with a purely local model. Finally, we will create a pair of ML models, which will allow us to evaluate the adequacy of the ML approach for the semantic parsing task. The results here presented were published as Meza-Ruiz et al. [2008a].

The outline of the chapter is as follows. In section 5.1 we will present some aspects of the implementation of HVS. This section starts by presenting the steps necessary to perform Spoken Language Understanding (SLU). In section 5.2 we will present a local discriminative model based on MaxEnt. In section 5.3 we will present the models formulated for the ML framework: a local model and a global model. In section 5.4 we will present a description of the experiments we perform, and the results obtained with them. Finally, section 5.5 will present a discussion of the significance of the results.

5.1 Hidden Vector State model

In this section we present the main considerations that should be taken into account when performing SLU with the HVS model. He and Young [2005] used the HVS

model for semantic processing, aiming to learn the mapping from natural language strings to semantic trees. The nodes in a semantic tree are semantic concepts, and its leaves are the words of the utterance. If we can guarantee that these trees have the property of being left branching then the HVS framework can model the semantic trees as a sequence of stacks. Semantic concepts can be terminal concepts, which means they are attached to a value of a slot. Figure 4.2 represents a semantic tree for the utterance *what flights are there arriving in chicago on continental_airlines after 2300*. This semantic tree also contains the representations of the slot-values for this utterance. The root of the tree represents the value for the `goal` slot. The rest of the slots are the sections of the paths which go from the root to the terminal concepts (here represented in lower-case). The values of these slots correspond to the leaves attached to the terminal concept. In this example, the slot-values represented in the semantic tree are:

- `GOAL=FLIGHT`
- `TOLOC.CITY_NAME=chicago`
- `AIRLINE_NAME=continental_airlines`
- `ARRIVE_TIME.TIME_RELATIVE=after`
- `ARRIVE_TIME.TIME=n2300`

Note that the path `ARRIVE_TIME.AIRLINE_NAME` is not a valid slot, but the `AIRLINE_NAME` sub-path is. Given a semantic tree, it is necessary to define a procedure to identify valid slots from invalid. We present some details in section B.1.

A semantic tree can easily be translated into a sequence of stacks for each word in the utterance. Figure 4.3 shows the sequence of stacks for our semantic tree example. In this example, it can be seen that the words which are not values are attached to a partial stack. We can identify these by the fact that their last element is not a terminal concept.

Since the stacks in the sequence satisfy the constraints of the HVS model, in which the transition from one stack to the following pops n states but only pushes one, the HVS model can be used to model the sequence of stacks. The slots and the partial slots become the vector states and the observations are the tokens of the utterance. In order to define an HVS model which will be used to perform SLU we need to identify:

1. The set of valid slots and their partial slots. These define the vector states of the HVS model.

2. The words that can be part of the utterances. These words define the alphabet of the HVS model.

3. The set of terminal concepts. This information is used to identify the values which are assigned to a slot.

These three elements can be identified from a training corpus.

5.1.1 Preprocessing

In practice, the HVS approach needs two preprocessing steps before it can be used. The first step is to prepare the input for training and labelling. The second step is to translate the slot values into semantic trees for training. The sequence of observations which forms the input for the HVS model is therefore modified in two ways:

- Add some extra observations which mark the start and ending of the observations (e.g., the symbols **start** and **end**). For instance our example becomes: ***start*** *what flights are there arriving in chicago on continental_airlines after 2300* ***end***.

- Rewrite some of the observations. This procedure is called lexical substitution. It replaces some words which belong together like *continental airlines* as *continental_airlines*, or it substitutes them with a label representing the class of the word like *airline_name*. See subsection 5.1.2 for more detail about this procedure.

In order to perform training it is necessary to translate the slot values into potential semantic trees. Subsection 5.1.3 presents the way in which we can generate such trees from the slot-values in a manner that reduces the search space for the parameter estimation process.

5.1.2 Lexical substitution

In the previous examples the compound noun *continental_airlines* was represented as a unique observation. This is a requirement for the HVS model as it has been defined, and it facilitates the recognition of values. However in practice the ASR module delivers compound words as two observations (i.e., *continental* and *airlines*). The HVS

model relies on a prepossessing step which will put these two words together. Actually HVS, as defined in He and Young [2005], goes further and replaces these words with a type class which represents them. This preprocessing step is called *lexical substitution*. He and Young [2005] do not limit lexical substitution to compound nouns but extend it to types of words. For instance, for our example the input would be modified into *what flights are there arriving in* `city_name` *on* `airline_name` *after* `time`.

The classes and their elements are defined in a gazetteer which is domain dependent. This gazetteer is used during lexical substitution to decide which sequences of words have to be replaced. As a default, the longest sequence of words is substituted with their first entry class, therefore shorter sequences are not substituted. However, this does not avoid some conflicts. For instance during our development we found that `new york` would be rewritten as `state_name` rather than `city_name`. Both substitutions are valid, but given the task the latter would be more appropriate. Therefore, we have to tune the gazetteer to have a preference for the latter substitution. Another common problem was that the gazetteer did not contain optional spellings, for instance `united` for *united airlines*. These variations had to be manually added to improve the performance.

This thesis concerns cases where such a gazetteer is not available, and therefore no tuning is required. The lexical substitution procedure still has to be applied to utterances, but in this case the objective is to group words which are potential values. This grouping is based on a training corpus. However, in the next chapter we will see how we can avoid this step.

5.1.3 Automatic constraint generation

In section 5.1 we presented the concept of semantic trees, which are the structures that HVS models learn to derive. However, when we come to train the model we do not have access to those structures, but only to slot-values. The whole semantic tree structure is hidden, but this is not a problem for the HVS framework. The partial information provided by the slot-values plus the constraints of popping *n* concepts and pushing one concept are used in the parameter estimation to propose all possible trees which yield the slot-values we are interested in. However, to consider all these trees is inadvisable as many of them are not related at all to the slot values we are interested in (e.g., for our example in Figure 4.3, we would like to avoid stacks with concepts for car rental). He and Young [2005] propose to constrain the number of possible trees

that the EM algorithm must take into account by visiting only the vector states which share elements with the slot-values. In this way, for the words which are values of a slot we only consider the vector state which represents its slot, and for the remaining words the vector states related with the slots values of the utterance.

In order to constrain the vector states during training He and Young [2005] introduce *abstract semantic representations* which encode the main elements of the slot values as partial semantic trees. Figure 5.1 shows the *abstract semantic representations* for our example. A disadvantage is that this representation must be proposed by the dialogue developer. He and Young [2005] propose the following procedure to identify a set of vector state constraints from the *abstract semantic representation* :

1. Flatten the *abstract semantic representation*. To flatten means to collect the stacks from root to leaves.

2. Split each flattened element into its partial stacks.

3. For each element in the set add a new vector state with a `DUMMY` element attached to it.

The result of this procedure is a set of valid vector states which constrains the vector states for the words in the utterance which are not a value of a slot. Table 5.1 presents the set of constraints for our example.

$$FLIGHT(TO\_LOC(CITY\_NAME),AIRPORT\_NAME,$$
$$ARRIVE\_TIME(TIME\_RELATIVE,TIME))$$

Figure 5.1: Example of *abstract semantic representation*

In this work we change the procedure suggested by He and Young [2005] in order to automatically generate these constraints from the slot-value annotations rather than relying on the abstract semantic representation. Instead of generating a whole set for the words which are not a value we generate individual sets for each word. These sets are based on the next slot which has to be generated. The procedure is shown in Algorithm 3. The constraint generation used in our experiments differs from the one suggested by He and Young [2005]. This version does not use the `DUMMY` concept. Table 5.2 shows the constraints generated by the Algorithm 3 procedure. A slightly modified version is used in our experiment, this is to account for the cases when two slots are next to each other and there is not a possible transition between them, this is

From step 1:	FLIGHT.TOLOC.CITY_NAME
	FLIGHT.AIRPORT_NAME
	FLIGHT.ARRIVE_TIME.TIME_RELATIVE
	FLIGHT.ARRIVE_TIME.TIME
From step 2:	FLIGHT.TOLOC
	FLIGHT
	FLIGHT.ARRIVE_TIME
From step 3:	FLIGHT.TOLOC.CITY_NAME.DUMMY
	FLIGHT.AIRPORT_NAME.DUMMY
	FLIGHT.ARRIVE_TIME.TIME_RELATIVE.DUMMY
	FLIGHT.ARRIVE_TIME.TIME.DUMMY
	FLIGHT.TOLOC.DUMMY
	FLIGHT.DUMMY
	FLIGHT.ARRIVE_TIME.DUMMY

Table 5.1: Example of generating constraints for vector state space using abstract semantic representations.

Algorithm 3 $constraintGeneration(utterance, slot-values)$

$prev \Leftarrow 0$
$constraints \Leftarrow [[goal]|$ word in utterance $]$
for slot in slot-values **do**
  $pos = get\_position(slot)$
  $constraint_{pos} \Leftarrow slot$
  $curr\_set \Leftarrow [slot]$
  for $partial\_slot$ in slot **do**
    $curr\_set.append(partial\_slot)$
  end for
  for $i = pos$ to $prev$ **do**
    $constraint_i \Leftarrow curr\_set$
  end for
  $prev \Leftarrow pos + 1$
end for

what	FLIGHT.TOLOC, FLIGHT
flight	FLIGHT.TOLOC, FLIGHT
are	FLIGHT.TOLOC, FLIGHT
there	FLIGHT.TOLOC, FLIGHT
arriving	FLIGHT.TOLOC, FLIGHT
in	FLIGHT.TOLOC, FLIGHT
chicago	FLIGHT.TOLOC.CITY_NAME
on	FLIGHT
continental_airlines	FLIGHT.AIRLINE_NAME
after	FLIGHT.ARRIVE_TIME.TIME_RELATIVE
n2300	FLIGHT.ARRIVE_TIME.TIME

Table 5.2: Example of automatic generated constraints for vector state space.

because they are not left-branching. We solve this by attaching the last concept of the current slot to the previous slot. Section B.2 explains the effects of this difference and its repercussions in performance between both implementations.

5.2 A discriminative model

In this section, we present a local discriminative model for the SLU task. As we have specified, a local model does not include dependencies between the hidden variables. Each hidden variable is chosen independently of other ones. This model provides a baseline for our following models. Since the HVS baseline was weak when compared with the state of the art performance (see Appendix B), this model will provide a stronger baseline, which is comparable with the state of the art.

To define this local model we use a Maximum Entropy (MaxEnt) method [Ratnaparkhi, 1999]. MaxEnt is a log-linear discriminative machine learning method which condition the outcome of a problem in terms of a set features of the observation. In the context of MaxEnt the nodes are not predicates, but labels, and the factors are feature functions instead of formulae.

The expression at the core of MaxEnt is presented in Equation 5.2.

$$p(y|\mathbf{x}) = \frac{1}{Z} exp\left(\sum_{i=1}^{n} \lambda_i f_i(y,\mathbf{x})\right) \quad (5.1)$$

where

$$Z = \sum_{y} exp\left(\sum_{i=1}^{n} \lambda_i f_i(y,\mathbf{x})\right)$$

The outcome for MaxEnt is a simple variable y rather than a set of variables Y. This is because, since each variable is independent of the rest of them, the model only considers one at the time.

The feature extraction functions f_i and the weights λ_i[1] are at the core of MaxEnt. The feature extraction functions signal the presence of a feature in the observations, for instance, take the feature extraction functions:

$$f_0(y,\S) = \begin{cases} 1 & \text{if } y \text{ is } FROM\_LOC.CITY \text{ and } orth \text{ is } chicago \\ 0 & \text{otherwise} \end{cases}$$

$$f_1(y,\S) = \begin{cases} 1 & \text{if } y \text{ is } FROM\_LOC.CYTY \text{ and } orth = car \\ 0 & \text{otherwise} \end{cases}$$

These functions check for some configuration within the observations § and the label y. For instance for f_0 it checks if the words has the orthography of *chicago* and that the label is $FROM\_LOC.CITY$ if that's the case then it would return 1, otherwise 0. On the other hand, the λ weights scale the feature. The intuition behind MaxEnt is that features like f_0 which could be common would have a larger weight than features as f_1 which is uncommon. In this way, features with larger weights have a higher probability.

A particular property of MaxEnt is the way it finds the weights λ. MaxEnt models all that is known and assumes nothing about that which is unknown. In other words, given a collection of training examples, MaxEnt choses a model which is consistent with all the facts, but is otherwise as uniform as possible.

To model the SLU task using MaxEnt when dealing with the slot-value representation is straight forward. We create two classifiers. One for the goal, and another one for the slots (see Figure 5.2). The goal classifier takes as observation the words and the bigrams of and utterance. For the slots, we create a classifier for each word i in the utterance (see Figure 5.3). The rest of the observations in the slot classifier are

[1] This is equivalent to the W weights in the ML framework.

extracted from the utterance. This observations are specific to a word *i* in the utterance. Therefore, the classifier decides the corresponding slot for that word. We introduce a new label to represent the absence of slot (i.e., `NONE`).

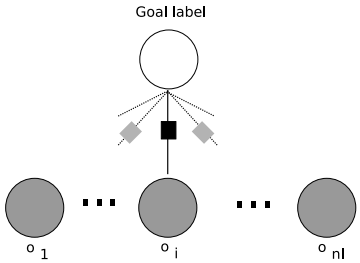


Figure 5.2: Factor graph for MaxEnt classifier for the goal

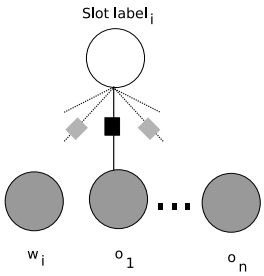


Figure 5.3: Factor graph for MaxEnt classifier for a slot

To label the slots of an utterance we process all possible windows for each word of the utterance. Then we collect the labels that are different to the one meaning absence of slot. We assign as a value the word to which this label belongs to.

5.2.1 Features for baseline

As specified before the feature extraction functions are at the core of the MaxEnt. This section present the features used in the MaxEnt baseline model. Because we wanted a fair comparison with the HVS model we only include features from the orthography of the words. The features used for the goal classifier are:

1. Orthography of each world in the utterance.

2. The bigrams of words present in the utterance.

The features used for the slot classifier:

1. Orthography of the current word.

2. Orthography of the previous two words.
3. Orthography of the following two words.
4. A flag if the words: *depart*, *arrive*, *departing* and *arriving*.

All features but the flags are standard features for classifiers in task as Name Entity Recognition or POS tagging task. We found the words for the flag features during development stage of our experiments. We noticed that this content words helped to distinguish departing and arriving slots. In the following chapter we will introduce a new set features based on syntactic chunks which extend this finding by adding a flag for each verbal or nominal chunk.

5.3 The ML models

In this section, we present the modelling done within the ML framework. In order to define a model in ML we start by defining the vocabulary, this is composed by the constants and predicates. In the context of the SLU task we need to specify relations between words of an utterance and the slots and values. For this reason we define the following constants:

- Positions for the words of an utterance, for instance: $1, 2, 3 \ldots$
- Orthography of the words, for instance: *what*,*chicago*.
- Slot labels, for instance: *TO_LOC.CITY*. Similar, to the MaxEnt model we add a label for the absence of slot (e.g., *NONE*).
- Goal label, for instance: *FLIGHT*.

With these constants we define the following predicates:

- *word/2* relates a word in a particular position with its orthography, for instance: *word(1,what)*, *word(6,chicago)*.
- *slot/2* relates a word in a particular position with its slot label, for instance: *slot(1,NONE)*, *slot(6,TO_LOC.CITY)*.
- *goal/1* specifies the goal label for an utterance, for instance *goal(FLIGHT)*.

In particular, *word/2* is the observable predicate, and *slot/2* and *goal/1* are hidden. With these definitions for the vocabulary we can define the local and global model for the SLU task.

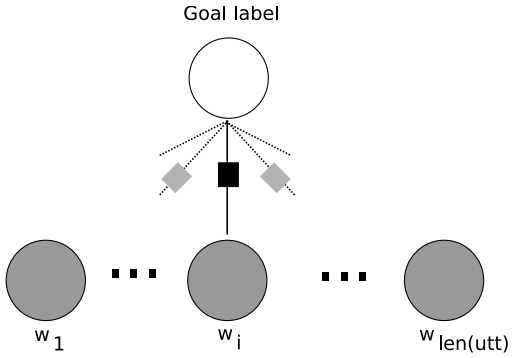


Figure 5.4: Factor graph for formula ϕ_{word}

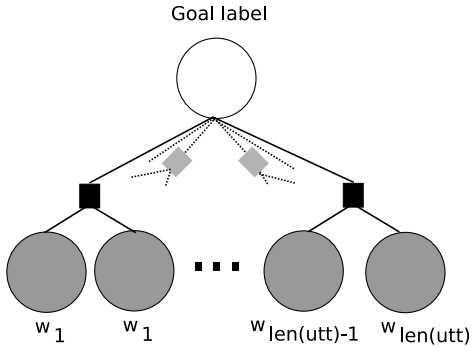


Figure 5.5: Factor graph for formula ϕ_{bigram}

5.3.1 Local model

Similarly to the MaxEnt model for the SLU task, we create two models, one for the goal and other for the slots. The local model in MLN consists of only local formulae. Basically, we can reproduce the features proposed for the baseline. In the case of the *goal* model we use the following formulae[2]:

$$\begin{aligned} \phi_{word} &\quad word(\_,+w) \wedge goal(g) \\ \phi_{bigram} &\quad word(p,+w_1) \wedge word(p+1,+w_2) \wedge goal(g) \end{aligned}$$

In this case, the formula ϕ_{word} relates the goal of an utterance with each one of the words in the utterance. This is similar to the strategy chosen in the MaxEnt model where the feature for the goal was the orthography of the words in the utterance. ϕ_{bigram} relates the bigrams present in the utterance with the goal. Figures 5.4 and 5.5 are the factor graphs for the goal formulae.

In the case of the *slots* model we can duplicate the MaxEnt model using the fol-

[2]The symbol "_" represents an undefined variable, each one can be identified with any name.

lowing formulae:

$$
\begin{aligned}
\phi_i &: word(p,+w) \wedge slot(p,+s) \\
\phi_{-1} &: word(p,\_) \wedge word(p-1,+w) \wedge slot(p,+s) \\
\phi_{-2} &: word(p,\_) \wedge word(p-2,+w) \wedge slot(p,+s) \\
\phi_{+1} &: word(p,\_) \wedge word(p+1,+w) \wedge slot(p,+s) \\
\phi_{+2} &: word(p,\_) \wedge word(p+2,+w) \wedge slot(p,+s) \\
\phi_{leave} &: word(p,\_) \wedge word(\_,leave) \wedge slot(p,+w) \\
\phi_{leaving} &: word(p,\_) \wedge word(\_,leaving) \wedge slot(p,+w) \\
\phi_{depart} &: word(p,\_) \wedge word(\_,depart) \wedge slot(p,+w) \\
\phi_{departing} &: word(p,\_) \wedge word(\_,departing) \wedge slot(p,+w)
\end{aligned}
$$

The formula ϕ_i relates a word with its slot which are in the same position. The formulae $\phi_{+/-n}$ relates a word in position $p+/-n$ with the slot in the position p. The formula $\phi_{leave|leaving|depart|departing}$ act as a flag in case a word with some of these orthographies is present in the utterance. These formulae are similar to the features defined in the MaxEnt baseline model. Figure 5.6 represents a graphical model for the slots model.

In these formulae we have extended the nomenclature of FOL with the sign $+$. We use this sign to signal the indexing for the weights. Recall that each formula ϕ is attached to a weight w. However, the MLN definition assumes that the formula is grounded, which is not the case for our model. The idea is to generate a weight for each grounded version of the formula. To have access to the right weight we indexed with some constants of a grounded formula. The sign $+$ signals the variables for which constants will be used as indices. For instance, for the formula ϕ_i there is a weight $w_{(chicago,FROM\_LOC.CITY)}$ which corresponds to the grounded version of the formula $word(p,chicago) \wedge slot(p,FROM\_LOC.CITY)$.

5.3.2 Global Formulae

For this current vocabulary, we experiment with different global formulae. We present here the ones which provided better performance for the semantic processing task. We start with the Markov assumptions, from our experiments the first and second Markov assumption empirically proved to have a better performance [3]. In order to represent

[3] 3rd and 4th Markov assumptions show to decrease the performance, this is because the pair slots of the formula are more sparse.

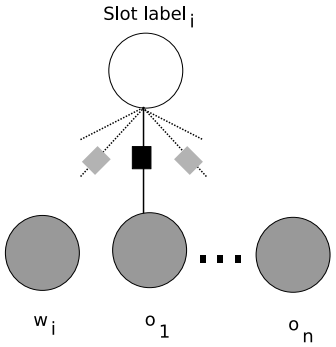


Figure 5.6: Model for slots as a single label.

them we use the following formulae:

$$-\phi_{1^{st}} \quad : \quad word(p,\_) \wedge word(p+1,\_) \wedge slot(p,+s_1) \wedge slot(p+1,+s_2)$$
$$-\phi_{2^{nd}} \quad : \quad word(p,\_) \wedge word(p+2,\_) \wedge slot(p,+s_1) \wedge slot(p+2,+s_2)$$

Formula $\phi_{1^{st}}$ is the first order Markov assumption. It correlates a slot in a position p with the slot in the position $p+1$. In a similar way formula $\phi_{2^{nd}}$ relates two slots separated by two positions. For these formulae, we restrict the weights to be negative. The overall effect of this choice is that these formulae act in a corrective way. Therefore, when the CPI algorithm finds a pair of unusual slots they get punished by discounting their corresponding weight. To define this formula in those terms helps with the implementation of the CPI algorithm. Figures 5.7 and 5.8 shows the factor graphs for both formulae.

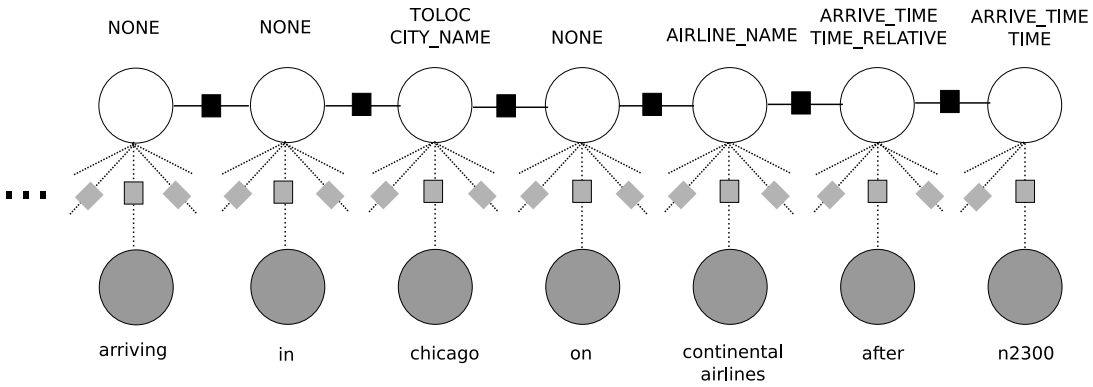


Figure 5.7: Factor graph for 1^{st} order Markov assumption

The Markov assumptions are global formulae, however they are not made to capture long distance dependencies. They capture dependencies in the vicinity of a slot.

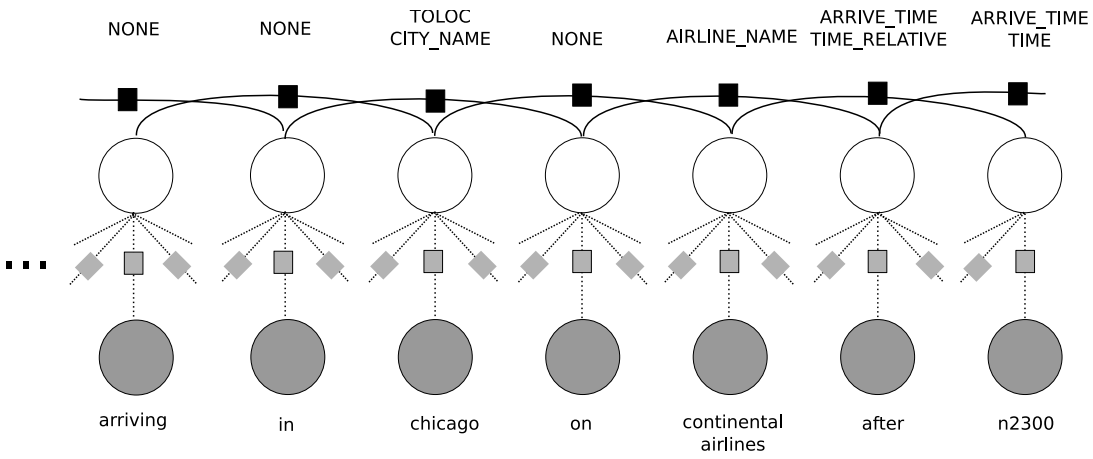


Figure 5.8: factor graph for 2^{nd} order Markov assumption

The following formula capture dependencies between slots which are further away than two slots, but not more than 6[4]:

$$-\phi_{3:6} \quad : \quad word(p,\_) \wedge word(p+i,\_) \wedge slot(p,+s_1) \wedge slot(p+i,+s_2)$$
$$\wedge 2 < i < 7 \wedge s \neq NONE$$

Figure 5.9 shows the factor graph for this formula for the case of the slot of the last word.

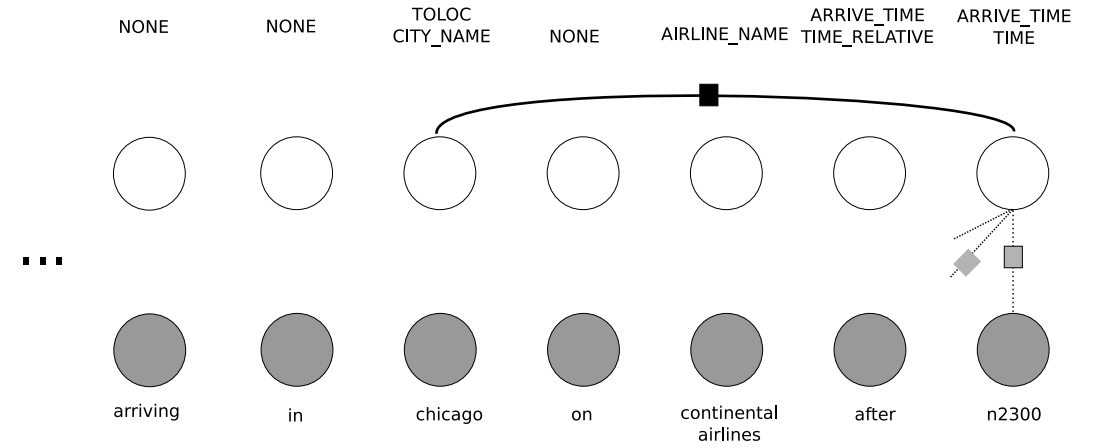


Figure 5.9: Factor graph for formula $\phi_{3:6}$

[4]Different distances parameters were tried this configuration was the one with better performance.

	Precision	Recall	F-score
ATIS3 $NOV-93$	N/A	N/A	90.30%
ATIS3 $DEC-94$	N/A	N/A	91.90%
ATIS3	88.75%	89.82%	89.28%
Communicator	87.20%	91.90%	89.50%

Table 5.3: Global scores reported in He and Young [2006]

5.4 Experiments and results

In order to make directly comparable the experiments in this chapter we need a similar setup for the three approaches. In particular we pay attention to two aspects:

- We use the same semantic representation for training and as outcome. In particular for the experiments of this chapter this semantic representation is slot-values[5].
- We use the same preprocessing steps for all approaches. In particular, the lexical substitution (see subsection 5.1.2).

The overall goal of this chapter is to compare the ML approach with a state of the art baseline for dialogue system. As specified before we have chosen the HVS approach as a baseline, given that is the state of the art approach. For this reason we created our own implementation of the HVS model. Table 5.3 reproduces the results presented by He and Young [2006] for the ATIS and Communicator corpora. This table shows the results for the three configurations of the ATIS corpora. The two first correspond to the best reported results. Because these results did not include the precision and recall measurement, for this reason we present the third results for the ATIS corpus where both test sets, *NOV-93* and *DEC-94*, are evaluated together but it includes the missing measurements. In these experiments the `goal` slot was predicted using a Tree-Augmented Naive Bayes network [He and Young, 2006]. The rest of the slots were predicted by a HVS model.

In the first set of experiments we use the HVS approach with minimal resources. In particular we omit the use of the gazetteer (subsection 5.4.1). The results of these experiments are used as our first baseline. The results to validate our implementation of

[5]This is slightly different from the experiments presented by He and Young [2005] which uses an extra labeling.

the HVS experiments were not satisfactory for the corpora (see Appendix B). Because this baseline was not strong enough we created a second baseline with a local discriminative approach. In this case, we use a MaxEnt classifier (section 5.2). This baseline also allow us to have a baseline of a local discriminative model, and it is higher than our HVS baseline.

In the final set of experiments, we test a ML model for the SLU task. For this, we create a local model which is equivalent to the discriminative baseline, and a global model. We use these models to perform semantic processing and with these experiments we try show the measure the adequacy of the ML for the SLU task (subsection 5.4.3). In summary, these are the experiments we perform in this section are:

1. HVS baseline with limited resources.
2. Discriminative local baseline with limited resources.
3. ML local model with limited resources.
4. ML global model with limited resources.

5.4.1 HVS Baseline experiments

The goal of implementing our own version of the HVS model was to test the adequacy of the approach in an environment with limited resources. For this reason, in these experiments we did not use the gazetteer in the lexical substitution procedure. In this set up, this procedure only puts together observations which appear as a value in the training corpora.

As expected, these results are worse than the state of the art results, since they do not include the semantic class information. Table 5.4 shows our results from these experiments. An interesting outcome of these results is that the results for the ATIS corpus are not statistically significantly different from the case when we use the gazetteer (see Appendix B). This was not the case for the Communicator corpus, in which the gazetteer helped with unknown words. This is the behaviour we expected since the gazetteer helped to identify potential values. For the rest of the chapter we consider this our first baseline. Although we will keep in mind that this is a weak baseline since our implementation did not exactly reproduce the results reported in literature when including the gazetteer (particularly we use auto constraint generation instead of the abstract semantic representation).

	Precision	Recall	F-score
ATIS-3 $NOV93$			
Global	85.37%	77.81%	**81.41**%
Average	81.75%	74.14%	77.03%
Exact match	46.12%	41.07%	43.45%
Goal Acc	79.24%		
ATIS-3 $DEC94$			
Global	86.67%	79.26%	**82.80**%
Average	83.39%	77.82%	80.10%
Exact match	46.37%	44.49%	45.41%
Goal Acc	80.22%		
Communicator			
Global	82.82%	82.63%	**82.72**%
Average	82.72%	83.29%	82.72%
Exact match	68.11%	66.95%	67.52%
Goal Acc	87.43%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table 5.4: Scores for the HVS model with limited resources on the ATIS $NOV93$ and $DEC94$ corpora and Communicator corpus.

	Precision	Recall	F-score
ATIS $NOV93$			
Global	85.86%	85.46%	**85.66**%
Average	81.08%	81.44%	80.82%
Exact match	63.84%	62.28%	63.06%
Goal Acc	79.69%		
ATIS $DEC94$			
Global	86.47%	86.01%	**86.24**%
Average	85.04%	85.15%	84.60%
Exact match	64.75%	63.15%	63.94%
Goal Acc	80.45%		
Communicator			
Global	87.69%	87.57%	**87.63**%
Average	85.98%	86.24%	85.59%
Exact match	77.58%	76.13%	76.84%
Goal Acc	80.45%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table 5.5: Scores for the MaxEnt (local discriminative) model with limited resources on the ATIS $NOV93$ and $DEC94$ corpora and Communicator corpus.

5.4.2 Discriminative baseline

Table Table 5.5 shows our results for the MaxEnt model. We can tell that we have closer results to the state of the art performance than our HVS results. These results are our discriminative baseline for the case of limited resources. In appendix B we present the results of this model when lexical substitution is used as part of the setting for the experiment.

	Precision	Recall	F-score
ATIS $NOV93$			
Global	91.21%	91.00%	**91.10**%
Average	85.51%	86.36%	85.48%
Exact match	77.24%	75.00%	76.10%
Goal Acc	90.84%		
ATIS $DEC94$			
Global	91.65%	92.44%	**92.04**%
Average	90.61%	91.34%	90.61%
Exact match	76.02%	75.51%	75.76%
Goal Acc	89.43%		
Communicator			
Global	89.90%	90.80%	**90.35**%
Average	89.07%	89.72%	88.96%
Exact match	81.96%	81.82%	81.89%
Goal Acc	88.61%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table 5.6: Scores for the local ML model with limited resources on the ATIS $NOV93$ and $DEC94$ corpora and Communicator corpus.

5.4.3 ML model with limited resources

We implemented the local and global model described in the previous sections and evaluate it for the minimal resources setup. Table 5.6 presents for the local model, and the Table 5.7 the global case.

We first notice that both local and global ML models produce better results than the baselines, and than the state-of-the-art results. These results were obtained without the use of lexical classes. We also notice that there is not much difference between the performance between the local and global models. We hypothesise this is in part because the global information has many influence over lexical tokens, and since these have been put together by the lexical substitutions during the preprocessing step, they

	Precision	Recall	F-score
ATIS $NOV93$			
Global	93.88%	88.94%	**91.34**%
Average	85.45%	84.42%	84.83%
Exact match	76.54%	72.10%	74.25%
Goal Acc	90.84%		
ATIS $DEC94$			
Global	94.51%	90.12%	**92.26**%
Average	91.97%	88.87%	89.79%
Exact match	72.27%	71.46%	71.86%
Goal Acc	89.43%		
Communicator			
Global	92.25%	90.53%	**91.38**%
Average	90.10%	89.52%	89.31%
Exact match	84.12%	82.84%	83.48%
Goal Acc	88.61%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table 5.7: Scores for the global ML model with limited resources on the ATIS $NOV93$ and $DEC94$ corpora and Communicator corpus.

have a great influence on the slots. In the following chapter we will explore the case where the preprocessing step does not take place and will observe if there is more interaction between the slots labels, which would made for more interesting models.

5.5 Discussion

In this chapter we have first introduced the baseline results for our experiments with the Markov Logic [ML, Richardson and Domingos, 2006] framework. These baseline results were generated using the Hidden Vector State [HVS, He and Young, 2005] and local discriminative clasifier [MaxEnt, Ratnaparkhi, 1999]. Additionally, we tested the ML under the same circumstances. The results that we have presented in this chapter are important to us because:

- The state of the art approaches, here represented by the HVS approach, have not been tested before under these circumstances (i.e. without the gazetteer).
- Some of the metrics we use in this work were not reported previously such as exact match and average score.
- We could test the performance of the ML approach in relation to the baseline.

As we mentioned in the introduction of this chapter, a baseline using the HVS model is acceptable because it does not include the concept of global feature which is a central property of the models we will present in the following chapters. These will allow us to contrast results with this baseline.

Once we established our baselines, we created the first model of ML for semantic processing with limited resources. We created two versions of this model, the first a local model equivalent to the MaxEnt model (see Table 5.6). In the second one, we add global dependencies (see Table 5.7). The local model outperforms our baselines at least by approximately 8% for the HVS model 5% for the MaxEnt model. The global model outperforms the local model in the Global measurement by 0.24% (with $\rho < 0.05$). In comparison with the state of the art results both models (local and global) do slightly better. However in the ATIS corpora, for Average and Exact Match metrics, the global does worse than the local. This result was unexpected since we hypothesised that the addition of global dependencies would produce better slot-values. This is not the case for the current state of the model. We do not have a definitive answer for this situation, we found that the global model produces better sequences of departure

and arrive times, however it introduces other errors. In the following chapter we will continue exploring the differences between global and local model in order to gain some light in the inner workings of such models.

In Figures 5.10, 5.11 and 5.12 we see a comparison of the different F-scores. For each metric we have results for the models we created. *HVS-Ori*[6] are the results reported by He and Young [2006], *HVS-rep* are the results for the replication experiments using the HVS approach (see Appendix B for a full description of these results). *HVS-lim* are the results for the HVS with limited resources. *ME-rep* and *ME-lim* are the results of the local discriminative model using MaxEnt for the replication and baseline with limited resources. *MLN-local* are the results for the local MLN model and the *MLN-global* are the results for the global MLN model. In this comparison we notice that we get better results with the MaxEnt replication models. However, remember that those results are obtained using the gazetteer. We observe that the global models are lower than the local for the average and exact match metrics. We hypothesise that the choice of predicates and the relations we can describe with them do not help to capture global dependencies. In the next chapter, we propose new predicates to take advantage of these dependencies. In general, the MLN models have a good performance considering they do not use the gazetteer information nor abstract semantic representations.

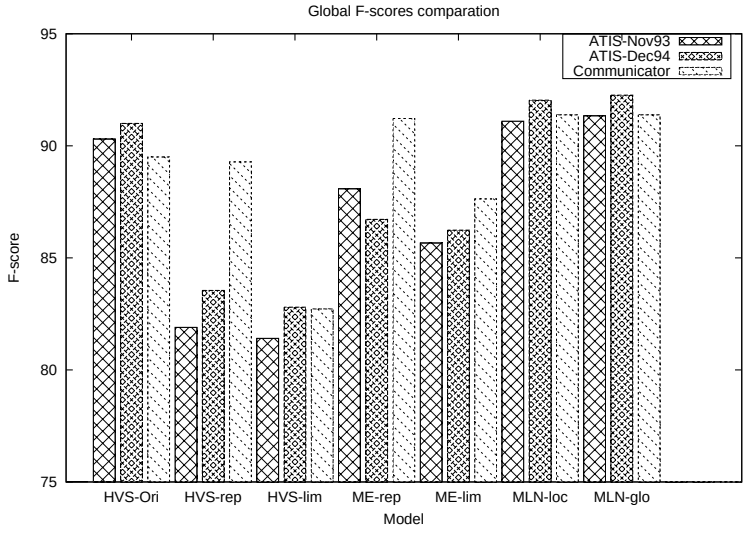


Figure 5.10: Global F-scores for different models

[6]There are only global results for the $HVS-ori$ model.

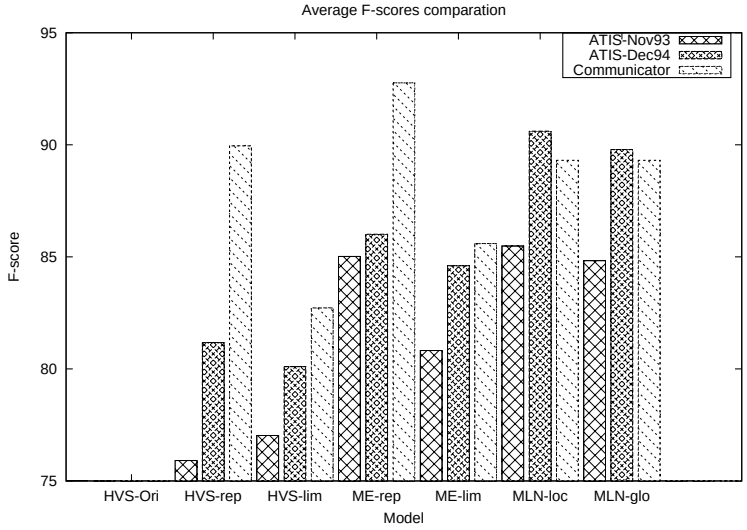


Figure 5.11: Average F-scores for different models

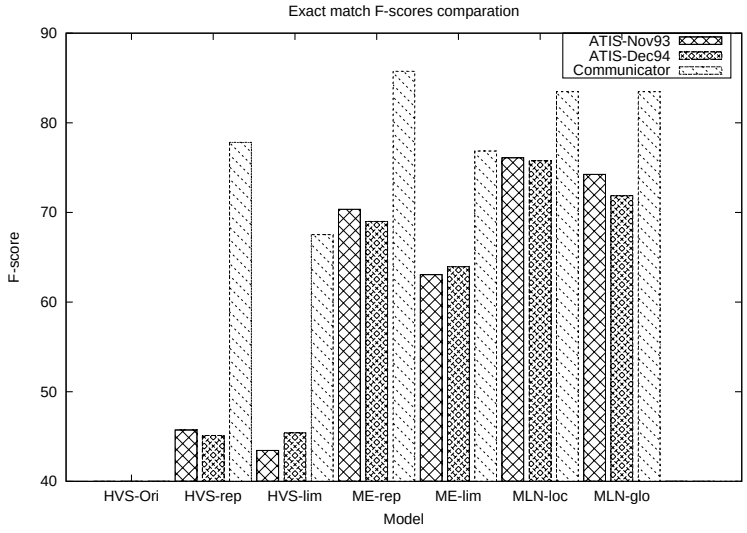


Figure 5.12: Exact match F-scores for different models

	Training	Testing
Previous models		
HVS-rep	55m	8m
HVS-lim	1h 24m	11m
MaxEnt-rep	22m	1m
MaxEnt-lim	19m	1m
MLN-loc	24m	1m
MLN-glo	55m	4m

Table 5.8: Training and testing times for different models. The training times were obtained using the ATIS corpus. The testing were obtained on the $NOV93$, $DEC94$ and Communicator corpora.

Finally, Table 5.8[7] presents the time taken by the models. The slowest model was the HVS model, while discriminative models were reasonably fast. These times show ML to be a viable option for semantic processing in a dialogue system.

In the following chapter we focus on changing the model to exploit better the global dependencies. In that chapter we also replace the lexical substitution, to be included as a part of the model.

[7]This times were obtained using an Intel(R) Pentium(R) 4 CPU 3.00GHz computer with 1GB of RAM memory.

Chapter 6

Extending the Markov Logic models for SLU

In the previous chapter we showed that it is possible to reach state of the art performance for semantic parsing in a dialogue system using the Markov Logic framework [ML, Richardson and Domingos, 2006]. This can take place with the minimum use of resources. However, some questions were still left open and we will address those in this chapter:

- Is it possible to avoid lexical substitution as a preprocessing step ? In order allow a fair comparison between the models of the previous chapter, we note that the previous ML model relied on lexical substitution to put together certain words into a single token (e.g., *continental airlines*). In this chapter we propose a model which does not have such requirements.

- Is there any other way to model the task in the Markov Logic framework? In this chapter, we explore new ML architectures for modelling the semantic processing task.

- What happens when we add extra lexical and syntactic information? In the previous chapter, we presented a case for the development of a Spoken Language Understanding (SLU) module using minimal resources. In this chapter, we analyse how to incorporate new sources of information into the ML models. In particular, we focus on a case where the source of such resources is off-the-shelf (e.g., POS tagger and syntactic chunker).

An early version of the results here presented were published as Meza-Ruiz et al. [2008b].

The outline of this chapter is as follows. Section 6.1 presents the BIO notation which is used to avoid the lexical substitution step. Section 6.2 introduces the two types of extensions we make to our previous model. The first one consists of modifying the architecture of the ML network that was described by the model (see section 6.2.1), and the second involves the use of extra information (see section 6.2.2). Section 6.3 presents the experiments we perform using both types of modifications. Section 6.4 presents the results of those experiments, along with an analysis of the performance of ML framework. Finally, section 6.5 presents a discussion of our findings.

6.1 BIO notation

The BIO notation first was introduced in the context of syntactic chunking [Ramshaw and Marcus, 1995]. In chunking, each word is labelled as being at the Beginning (**B**), Inside (**I**) or Outside (**O**) of a chunk. We have adapted this notation to allow it to be used for the slots of the frame-based semantic representation. The intuition behind using the BIO notation is to capture when a word represents a value of a slot and when it does not. With this information we aim to model that certain words are values and should be associated with a slot. This notation fits with the slot-value representation because all values are a sequence of words of the utterance.

In order to adapt the BIO tags to the frame-based semantic representation we assign the tag **O** to the words which are not part of a value of a slot. If the word is part of a value, we assign the tag **B** when the word is the starting word of the value, otherwise we assign the tag **I**. With these assignations we are able to transform the ATIS and Communicator corpora to include BIO tags. No extra resources are necessary and it is done directly from the slot labelling. Furthermore, this transformation makes it possible to model the semantic processing task without relying on lexical substitution as a preprocessing step (see subsection 5.1.2).

Figure 6.1 presents an example of this assignation for our example utterance from the ATIS corpus – the BIO tag assigned to each word is after this in bold font. There are 4 **B** tags which correspond to the four first words of values of the slots. There is only one **I** tag which arises, from the word *airlines* which is the second word in the value of the slot `AIRLINE_NAME`. Figure 6.2 presents an graphical version of the assignation on the BIO labels.

*what (**O**) flights (**O**) are (**O**) there (**O**) arriving (**O**) in (**O**) chicago (**B**) on (**O**) continental (**B**) airlines (**I**) after (**B**) n2300 (**B**)*

```
GOAL = flight
TOLOC.CITY_NAME=chicago
AIRLINE_NAME=continental airlines
ARRIVE_TIME.TIME_RELATIVE=after
ARRIVE_TIME.TIME=n2300
```

Figure 6.1: Example of BIO tag assignment for utterances from the ATIS-3 corpus. Each word of the utterance is followed by the BIO tag between parentheses. The frame-based semantic representation defines the BIO labels.

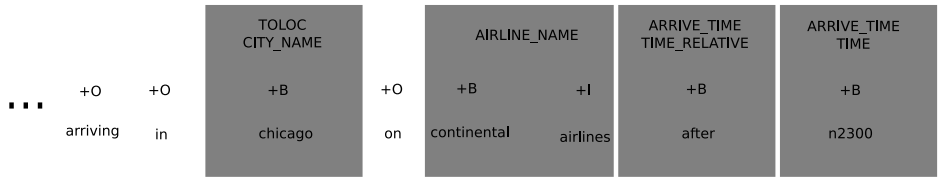


Figure 6.2: Example of BIO slot assignation. The first value inside a slot are **B**, the rest of values inside a slot are **I** and the words outside the slot are **O**.

6.2 Extension to the Markov Logic model

In this section we present the extensions that are added to the Markov Logic (ML) models which were described in Chapter 5. Extending ML models is relatively easy: we simply add new predicates and formulae to them. In this case we will extend the model in two directions:

1. Modify the architecture of the hidden variables.

2. Add extra labellings to the information available.

With the first modification we capture the dependencies between the elements of the semantic representation: for instance, between a word being a starting position of a value and its slot. For the second modification, we will focus in measuring the benefits of including complementary information in the system. Both of these modifications are orthogonal, meaning that we can use the extensions of the first modification along

with the modifications of the second.

Although, earlier we advocated models which do not rely on the use of extra information, we also want to test how well the ML is able to incorporate extra information. Besides the gazetteer of classes used in the original lexical substitution, we will look into using state of the art NLP systems. We use a POS tagger and a syntactic chunker to automatically create extra labellings. In this way these extra labellings can be automatically generated and there is no need to dedicate extra human resources to this task. The extra predicates and formulae are presented in subsection 6.2.2.

6.2.1 Models

The following extensions to the model have the goals of, firstly incorporating the BIO labelling, and secondly, making explicit some of the relations between the elements of the slots. In order to incorporate the BIO notation into the model, the first strategy is to modify the slot labels: these are the slot constants defined in section 5.3. We will call this model a *one layer* model (subsubsection 6.2.1.1). The second modification is to define a predicate that is specific to the BIO notation. In this case we will define a model with multi-predicates. We will call this model a *two layer* model (see subsection 6.2.1.2).

6.2.1.1 One layer model

This model uses the new type of constant which we will call *BIOSlot*. This type represents the concatenation of slot label and BIO tag. For instance `TOLOC.CITY_NAME+B` is a new constant for *BIOSlot* where `TOLOC.CITY_NAME` is a slot label and `B` is the tag which indicates the beginning of a slot. This is the main extension for this model. The intuition behind it is to incorporate the BIO notation in the easiest way possible. A similar strategy can be employed to allow us to use an off-the-shelf classifier, such as MaxEnt.

The formulae of the *one layer* model consists of all the formulae introduced in the previous chapter (see section 5.3), but with the replacement of the *slot/2* hidden predicate with the hidden predicate *bio-slot/2*. This predicate encodes the slot which is label concatenated to its BIO tag for a word in a specific position. For instance, the predicate *bio-slot(7,FROMLOC.CITY_NAME+B)* represents the slot and BIO tag for the word *chicago* in example 6.1.

The following is an example of the substitution of a predicate *slot* by a *bio-slot*:

$$(original)\phi_i \quad : \quad word(p,+w) \wedge slot(p,+s)$$
$$(new)\phi_i \quad : \quad word(p,+w) \wedge bio\_slot(p,+s)$$

We can do this for all the formulae introduced in the previous chapter. With this modification, we have a model which captures the sequence of the slot labels and the BIO tags that are attached to them.

Since the new constants encode when words are parts of values, lexical substitution is not needed anymore. Figure 6.3 shows the MLN for our example utterance using the original model, compared with that produced by the *one layer* model. The nodes in gray represent the observed predicate *word/2* and the nodes in white represent the hidden predicates *slot/2* for the original Markov network and *bio-slot/2* for the new one. Notice that the original model is shorter than the *one layer* model, since the words which are values are not rewritten (i.e., *continental airlines* is reproduced as *continental_airlines*). The *one layer* model captures the possibility that a set of consequent words are a value rather than relying on a preprocessing step.

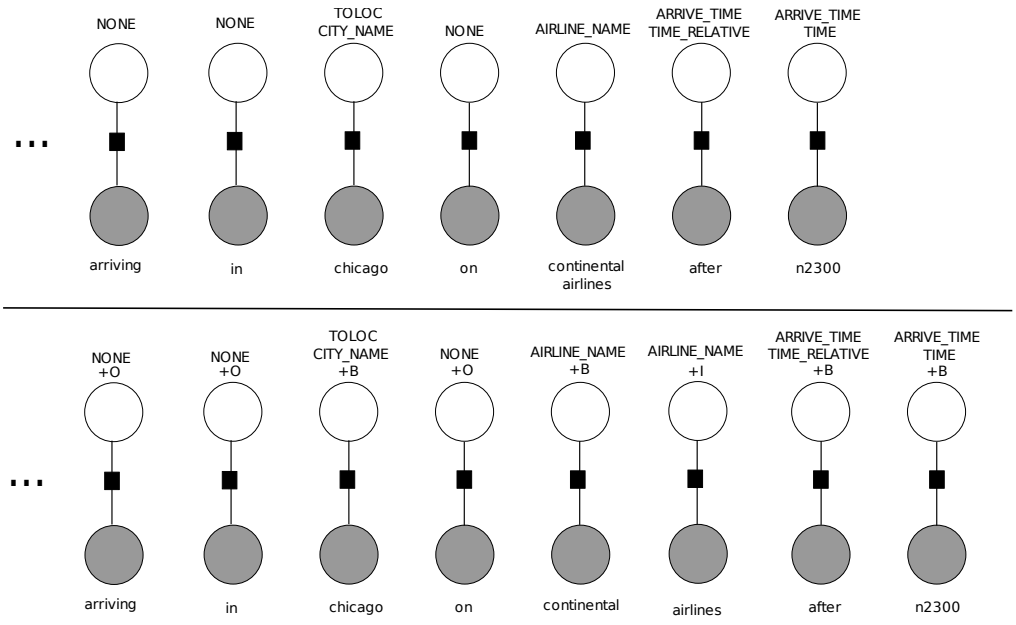


Figure 6.3: Comparison of Markov networks for the original model and *one layer* model. The original Markov network uses the slot labels of the original model. The *one layer* model uses labels which concatenate the slot with the BIO tag.

6.2.1.2 Two layer model

For the *two layer* model we split the definition of the type *BIOSlot* into two types: the type *Slot*, in which its constants are all possible slot labels, and the type *BIO*m in which its constants are the BIO tags. The type *Slot* is identical to the use in the model in the previous chapter.

In the *two layer* model, we use three hidden predicates:

- *slot/2* is identical to that used in the previous chapter. For instance, *slot(7, TOLOC.CITY_NAME)* for the word *chicago* in our example in Figure 6.1.
- *bio/2* encodes the BIO tag for a word in a certain position, for instance, *bio(7,B)* for the word *chicago* in our example in Figure 6.1.
- *bio-slot/3* encodes the slot label and the BIO tag for a word in a certain position, for instance, *bio-slot(7,TOLOC.CITY_NAME,B)* for the word *chicago* in our example in Figure 6.1. This predicate is equivalent to the concatenation for the slot label and the BIO tag introduced in the one layer model.

Figure 6.4 shows the *two layer* model Markov network for our example utterance. The reason this model is called *two layer* is because it has two main layers of predicates: a first layer for the predicate *slot/2* (first sequence of small white nodes) and a second for the predicate *bio/2* (second sequence of small white nodes). The *bio-slot/3* predicate joins both layers into one (larger white nodes). In this case, we represent the factors which links each predicate to the word (small gray node) in the same position. The intuition behind this model is to replicate the *one layer* model by using the *bio-slot/3* predicate. However, we extend this model by capturing the relationship that the slots and the BIO tags have with the utterance by using the *slot* and *bio* predicates.

The use of multiple hidden predicates in a model requires a way to ensure that the predicates are compatible between them. For instance, if we have the ground predicate *bio-slot(7,TOLOC.CITY_NAME,B)* we need to ensure that the predicate *slot/2* is grounded to *slot(7,TOLOC.CITY_NAME)* and the predicate *bio/2* to *bio(7,B)*. To do this, we use a set of hard formulae which we call *structural* constraints. These formu-

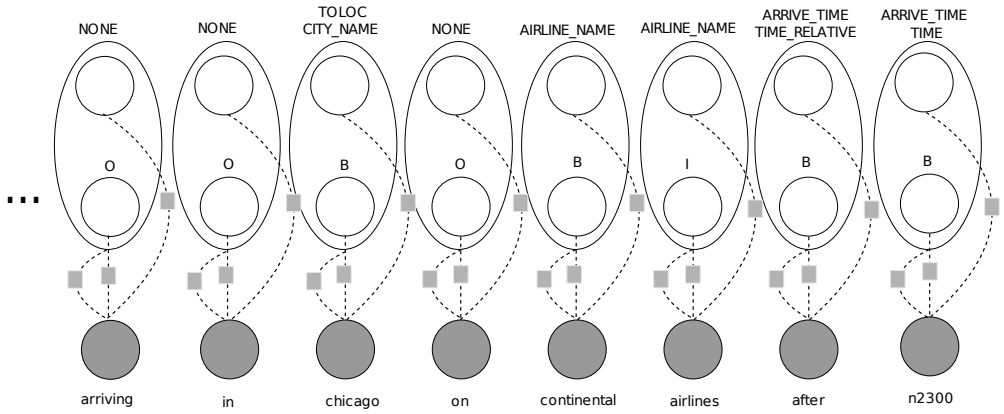


Figure 6.4: Two layer Markov Network using the *slot/2* predicate (first layer of white nodes), *bio/2* predicate (second layer of white nodes) and *bio-slot/3* (nodes which join the two previous predicates).

lae are presented here:

$$\begin{aligned}
\phi_{sc_1} &= bio-slot(i,_,b) \Rightarrow bio(i,b) \\
\phi_{sc_2} &= bio-slot(i,s,_) \Rightarrow slot(i,s) \\
\phi_{sc_3} &= bio(i,b) \Rightarrow \exists s.bio-slot(i,s,b) \\
\phi_{sc_4} &= slot(i,s) \Rightarrow \exists b.bio-slot(i,s,b)
\end{aligned}$$

Formulae ϕ_{sc_1} and ϕ_{cs_3} ensure that if the ground predicates *bio-slot/3* are *bio/2* are true, they have the same BIO tag b. The formulae ϕ_{sc_2} and ϕ_{cs_4} ensure a similar condition for the slot s.

6.2.2 Extra resources

Orthogonally to the extension to the architecture of the ML model, we extend the information available to the ML network on three levels:

- *Part of speech tags:* We extract the POS tags for all the words belonging to the utterances. For this we use the *TnT* POS tagger [Brants, 2000].
- *Syntactic chunks:* We divide the utterances into syntactic chunks. For this we use the *CASS* chunker [Abney, 1996].
- *Lexical classes:* We assign classes to the words which belong to a syntactic class. For this we use the classes defined in the Database of the ATIS system.

The *TnT* POS tagger was chosen both because it had been used before with the ATIS corpora, and because a setting to produce the POS tags was available via the ATIS corpus. The *CASS* syntactic chunker was chosen because it provided good results without the necessity of setting a training stage, since it is rule based.

In order to incorporate these labellings we defined the following ML predicates:

- *pos/2* relates a word in a particular position to its POS tag, for example *pos(1,wdt)* and *pos(6,nn)*.
- *chnk/2* relates a word in a particular position to the chunk label. For example *chnk(1,orc0)* signals that the word in position 1 is a head of a relative clause and *chnk(6,nx)* signals that the word in position 6 is a head of a noun phrase.
- *class/3* relates a multi-word starting in a particular position and ending in the same or another position to a class. For example *class(6,6,city)* signals that the word in the position 6 is a city. And *class(8,9,airline)* that the words from the 8th to the 9th position represent an airline.

For our models these predicates are observable, and we use them to define local formulae. For example, for the *pos/2* predicate we define the following formulae:

$$\begin{aligned}
\phi_{pos_i} &: pos(p,+w) \wedge slot(p,+s) \\
\phi_{pos_{-1}} &: pos(p,_) \wedge pos(p-1,+w) \wedge slot(p,+s) \\
\phi_{pos_{-2}} &: pos(p,_) \wedge pos(p-2,+w) \wedge slot(p,+s) \\
\phi_{pos_{+1}} &: pos(p,_) \wedge pos(p+1,+w) \wedge slot(p,+s) \\
\phi_{pos_{+2}} &: pos(p,_) \wedge pos(p+2,+w) \wedge slot(p,+s)
\end{aligned}$$

These formulae relate a slot with POS tags in a window from two previous to two following words. Additionally, for the *chnk/2* predicate we define the following formulae:

$$\begin{aligned}
\phi_{chnk_{nx}} &: word(p_1,_) \wedge word(p_2,+w) \wedge cass(p_2,nx) \wedge slot(p_1,+s) \\
\phi_{chnk_{vx}} &: word(p_1,_) \wedge word(p_2,+w) \wedge cass(p_2,vx) \wedge slot(p_1,+s)
\end{aligned}$$

These formulae look for the heads of phrases in the chunks and relate their orthography with the slot. With these formulae we attempt to extended the intuition behind the keywords used in the formulae ϕ_{leave}, $\phi_{leaving}$, ϕ_{depart} and $\phi_{departing}$. But in this case,

the words are automatically identified rather than proposed by the system developer. Finally, for the *class/3* predicate we define the following formulae:

$$\phi_{class_{start}} : word(p_1, \_) \wedge class(p_1, \_, +c) \wedge slot(p_1, +s)$$
$$\phi_{class_{end}} : word(p_1, \_) \wedge class(p_1, p_2, \_, +c) \wedge slot(q, +s) \wedge q \geq p_1 \wedge q \leq p_2$$

These formulae connect the lexical classes of a word to their slots.

6.3 Experiments

Our experiments address the three questions formulated in the introduction of this chapter. All the experiments in this chapter exclude the use of the lexical substitution as a preprocessing step. We have two different set of experiments, which attempt to measure the effect of the two ML models: the *one layer* and the *two layer* model. Additionally, we test for each type of model the effect of adding extra information into the ML models. This should allows us to answer the open questions presented at the start of this chapter.

6.3.1 One layer model

We start with a local version of the model. Since this model is the first that has been created without lexical substitution, we created an equivalent MaxEnt model to compare it with (in this case we extended our model to include a Gaussian prior, which produces a better MaxEnt model). In order to extend the model into a global model, we test the following conditions:

- Local model plus first order Markov assumption, formula ϕ_1^{st}.
- Local model plus first and second order Markov assumption, formulae ϕ_1^{st} and ϕ_2^{nd}.
- Local model plus first and second order Markov assumptions and long distances relation between slots separated by more than two positions but fewer than six positions, formulae ϕ_1^{st}, ϕ_2^{nd} and $\phi_{3:6}$.

We call this set of experiments the *global formulae* experiments.

Once we identify the best global results, we test the addition of extra information. For this we run experiments as follows:

- Adding formulae based on POS tags.
- Adding formulae based on syntactic chunks.
- Adding formulae based on POS tags and syntactic chunks.
- Adding formulae based on classes.
- Adding formulae based on POst tags, syntactic chunks and classes

We call these experiments the *extra information* experiments.

6.3.2 Two layer model

We start the experiments on the *two layer* model with a global model which is equivalent to the best formulae of the first layer model. However, the *two layer* model offers another dimension for experimentation: to include or exclude hidden predicates. So we perform the following experiments:

- Only using the *bio-slot/3* predicate.
- Using the *bio-slot/3* and the *slot/2* predicates.
- Using the *bio-slot/3* and the *bio/2* predicates.
- Full model.

We call these experiments the *multi-predicate* experiments.

Additionally, we apply the *extra information* experiments to the best model of the *multi-predicate* experiments.

6.4 Results

This section describes the results of the experiments described aboce. The experiments were done using the ATIS corpus [Dahl and et al., 1994] (see section 3.1) and the Communicator corpus [Bennett and Rudnicky, 2002] (see section 3.2). In this section, for each proposed experiment we only present the results for the combination of ATIS *NOV*93 and ATIS *DEC*94 testing corpora. This is because most of the experiments are exploratory in nature, examining the different model combinations. However in section 6.5 we present the results of our best model when used with the Communicator corpus and the two versions of the ATIS corpora.

Experiment	Global	Average	Exact match
Original local model	91.61%	88.04%	75.93%
Original global model	91.83%	87.10%	73.05%
MaxEnt	73.23%	68.91%	38.70%
Local	89.21%	85.90%	72.02%
$\phi_{\mathbf{1^{st}}}$	**90.09%**	**87.23%**	**73.55%**
$\phi_{1^{st}}$ $\phi_{2^{nd}}$	88.89%	85.45%	72.13%
$\phi_{1^{st}}$ $\phi_{2^{nd}}$ $\phi_{3:6}$	85.84%	82.72%	67.76%

Table 6.1: Results for the *global formulae* set of experiments for the *one layer* model (using *bio-slot/2*) on the ATIS corpora.

6.4.1 One layer model

Table 6.1 presents our results for the *global formulae* experiments. There are several observations, that can be made regarding these experiments. Firstly, the addition of the BIO notation in our models reduces the performance when compared with our original models from the previous chapter (which needed lexical substitution as a preprocessing step). This is because it is a harder task. Secondly, the local ML model outperforms the MaxEnt model, as was the case in previous experiments (see section 5.2). Thirdly, we reach competitive performance when we use global formulae in the ML model, particularly for the average and exact match metrics. Fourthly, it can be seen that the best performance for our current model occurs when only using the $\phi_{1^{st}}$ formula; this differs from our original model which performed better when additionally using the $\phi_{1^{st}}$ and $\phi_{3:6}$ formulae.

Table 6.2 shows the results for the *one layer* model when using extra information from our best model from the *global formulae* experiments. This model is a global model which only includes the $\phi_{1^{st}}$ formula. From these results, we can see that using POS tags does not necessarily help with the task (this may, however, be related to the quality of the POS tagger). On the other hand, using the syntactic chunks produces an improvement. The table also shows the results for the model which uses information from the gazetteer. The performance here increases by more than 2% for global scores, but it is also significantly improved for the exact match score which increases by 5%.

Experiment	Global	Average	Exact match
POS	89.76%	86.43%	71.71%
Chunk	**90.59**%	**87.59**%	**74.59**%
POS + Chunk	90.38%	87.47%	73.64%
Classes	92.60%	90.29%	80.43%
POS + Classes	92.37%	88.88%	78.19%
Chunk + Classes	**92.70**%	**90.39**%	**80.61**%
POS + Chunk + Classes	92.53%	90.11%	79.41%

Table 6.2: Results for the *extra information* set of experiments for the *one layer* model (using *bio-slot/2*) for the ATIS corpora.

Experiment	Global	Average	Exact match
bio-slot/3	89.73%	86.86%	72.52%
bio-slot/3 slot/2	89.00%	86.10%	72.35%
bio-slot/3 bio/2	**90.03**%	**87.35**%	**73.94**%
bio-slot/3 slot/2 bio/2	89.14%	85.45%	72.92%

Table 6.3: Results for the *multi-predicate* set of experiments for the *one layer* model (using *bio-slot/3* and *slot/2*) on the ATIS corpora.

6.4.2 Two layer model

Table 6.3 shows our results for the *multi-predicate* set of experiments for the *two layer* model. Recall that this model incorporates the first Markov assumption which was favourable for the *one layer* model. In this case, the best model resulted from using the *bio-slot* and *bio* predicates. When we compare these results with the results for the *one layer* model, we notice that there is not much difference between it and the directly comparable model (this is the global model with first Markov assumption, formulae $\phi_{1^{st}}$).

Table 6.4 shows our results when we add extra information to our best model of the *multi-predicate* experiments. We notice that again the most informative element is the lexical classes gazetteer. This is compatible with the *one layer* model findings. We also notice that the POS tag information was not helpful.

When comparing the *one layer* model with the *two layer* model for the *extra information* experiments, we found that when the models do not include the classes

Experiment	Global	Average	Exact match
POS	90.14%	85.95%	74.01%
Chunk	**90.70**%	**87.75**%	**75.08**%
POS + Chunk	89.71%	86.41%	71.40%
Classes	**92.74**%	**90.28**%	**80.14**%
POS + Classes	92.33%	89.68%	79.25%
Chunk + Classes	92.48%	89.72%	78.33%
POS + Chunk + Classes	92.96%	89.81%	79.32%

Table 6.4: Results for the *extra information* set of experiments for the *two layer* model (using *bio-slot/3* and *bio/2*) on the ATIS corpora.

information, the *two layer* model outperforms the *one layer* model. This is because the model is able to take advantage of the independent BIOi-labelling statistics. However, when we include the lexical classes, the *one layer* model performs slightly better than the *two layer*. This is because the BIO layer does not play as important a role here as in the other models, since the lexical class information contains the information when a potential value starts and ends. This information is encoded by the *class/3* predicate.

6.4.3 Learning behaviour with ML

Before we conclude, we can analyse the learning behaviour of the ML framework. In order to do this, we present the learning times for some of the models of in our experiments. We also include the learning curves for these systems. The experiments to be analysed are:

- Local model for *one layer* model (No extra information).
- Best global model for *one layer* model (No extra information).
- Best global model for *one layer*, using chunks.
- Best global model for *one layer*, using classes.
- Best global model for *two layer* model, this is *BIO/2* and *bio_slot/3*.
- Best global model for *two layer*, using chunks.
- Best global model for *two layer*, using classes.

We do not include the local model for the *two layer* model since it is multi-predicate. The local model requires that there are no links between the hidden predicates. In order to satisfy this. We would need to take out the structural constraints. The resulting model would not be functional, since it would not ensure that the semantic representations are well formed.

Table 6.5[1] presents the time for the models. Firstly, we can see that the longest time for testing is 5 minutes. This corresponds to parsing approximately $1,700$ utterances. Thus in the slowest case, it takes on average 0.17 seconds to find a semantic representation for the utterance. Secondly, during training we notice the effect of a making the ML model more complex. The *two layer* model is considerably slower than the *one layer* model. We see that the chunk formulae made the model slower. This is because this model requires a large amount of weight factors since it combines slots with any head word in the utterance.

Figure 6.5 shows the learning curves for the *one layer* and *two layer* models. In the case of the *one layer* model the learning curve shows there is not too much advantage for using a global model for small corpora. We notice that all global versions overfit the model. We attribute this to the number of words without slots. The $\phi_{1^{s_t}}$ formula helps to exclude sequences of invalid slots, so with a small corpus it learns to punish sequences of slots which have not seen previously in the corpora. As a result of this, it rewards sequences of *not-slots*, these are words which are not part of a value of a slot. This tendency disappears as soon as there are more than $2,500$ utterances in the data available. In the case of the *two layer* model, we notice that the learning curve is more informative. We can see that the *two layer model* with the classes is much better than the other models. This is because the *two layer* model exploits the relation between the lexical classes and the *bio/2* predicate. This configuration better captures when a word is potentially a value. The main gain in the development corpus is for the slots `FLIGHT_MOD`, `FROMLOC.CITY_NAME` and `TOLOC.CITY_NAME`. In particular, we notice that errors related to the concepts `FROMLOC` and `TOLOC` involve prepositions followed by unknown cities.

[1]This times were obtained using an Intel(R) Pentium(R) 4 CPU 3.00GHz computer with 1GB of RAM memory.

	Training	Testing
Previous models		
HVS	1h 24m	11m
MaxEnt	19m	1m
Local ML	24m	1m
Global ML	55m	4m
One layer model		
Local model	1h 5m	1m
Global model	1h 28m	2m
Global model + chunks	3h 46m	5m
Global model + classes	1h 22m	4m
Two layer model		
Global model	3h 2m	4m
Global model + chunks	5h 42m	4m
Global model + classes	2h 1m	2m

Table 6.5: Training and testing times for ML models. Each model consists of 20 training iterations and testing on the $NOV93$, $DEC94$, and both versions of the ATIS corpus. The training times were obtained using the ATIS corpus. The testing were obtained on the $NOV93$, $DEC94$ and Communicator corpora.

6.5 Discussion

In this chapter we have presented some modifications to the ML model presented in the previous chapter. These modifications address the open questions presented at the start of this chapter, which I repeat here:

1. Is it possible to avoid lexical substitution as a preprocessing step? Yes, it is possible. However, it comes with a reduction in the performance of the semantic processing. This can be addressed by incorporating more information into, or changing, the ML model.

2. Is there any other way to model the task in the Markov Logic framework? Yes, there is. In this chapter we showed two different models: a *one layer* and a *two layer* model. Both of them incorporate the BIO notation. In general terms, we notice that the *one layer* model is a good model. However, with the *two layer* model it is possible to reach a better exact match performance. However, the complexity of the *two layer* model increases the training and testing time.

3. What happens when we add extra lexical and syntactic information? We notice that we can increase the performance of the system. In particular, we found that our model can take advantage of the syntactic chunks, by performing better than a model which does not include this information. This was not the case for the POS tag labelling. We also noted that using the classes information provided by a gazetteer is the best way of improving the performance.

Table 6.6 shows the summary of our results for the different corpora used in this chapter. We notice that the reduction on performance compared to our previous model is significant for the ATIS *DEC*94 test and the Communicator corpora. We hypothesise that this happens because the corpora contain more multi-word values, however we cannot test this since this data is by its nature our testing data. In the case of the ATIS test corpora, we see an significant improvement with the Exact match metric. This means that both models produce semantic representations which are more coherent than our model of the previous chapter. This is despite the fact that the task is harder since in this chapter we omit the lexical substitution. We also notice that the best source of information is the classes: in all the experiments, the inclusion of this information allows the model to outperform any of the directly comparable models.

This concludes our experiments with the ATIS and Communicator corpora which began in the previous chapter. We have shown that the ML framework is adequate for

	Global	Average	Exact match
ATIS *NOV*93			
Previous result	91.34%	84.94%	74.25%
	One layer model		
Global	91.01%	86.99%	75.85%
Global + chunks	91.78%	87.51%	76.64%
Global + classes	92.56%	89.51%	81.00%
	Two layer model		
Global	91.02%	86.91%	75.14%
Global + chunks	91.35%	86.94%	77.03%
Global + classes	92.62%	88.81%	80.05%
ATIS *DEC*94			
Previous model	92.26%	89.79%	71.86%
	One layer model		
Global	89.28%	87.48%	71.25%
Global + chunks	89.50%	87.67%	72.54%
Global + classes	92.63%	91.07%	79.87%
	Two layer model		
Global	89.91%	87.81%	72.75%
Global + chunks	90.14%	88.57%	77.03%
Global + classes	92.85%	91.76%	80.14%
Communicator			
Previous model	91.38%	89.31%	83.48%
	One layer model		
Global	86.82%	84.53%	78.40%
Global + chunks	87.20%	85.02%	78.88%
Global + classes	87.20%	85.02%	78.88%
	Two layer model		
Global	87.36%	85.02%	78.54%
Global + chunks	87.29%	85.49%	79.22%
Global + classes	87.42%	85.40%	79.73%

Table 6.6: F-score of *one layer* and *two layer* models on the ATIS $NOV93$ and $DEC94$ corpora and Communicator corpus.

performing semantic processing for dialogue systems. We have demonstrated that under the setting of no extra resources, it can provide good performance. We also shown that it is possible to include extra information within our models, by adding extra predicates and formulae. This gives the ML framework the attractive characteristic of being able to be extended as soon as more information is available. However, we notice that when compared with current advances in the field, the ML model is still not producing the best performance [Zettlemoyer and Collins, 2007; Mairesse et al., To appear.]. Further research must be carried out to identify what elements are required in the ML framework in order to achieve maximum performance.

In the following chapter, we investigate the adequacy of the ML framework for a semantic representation that is more complex than frame-based semantic representations which are the ones used by the ATIS and Communicator corpora.

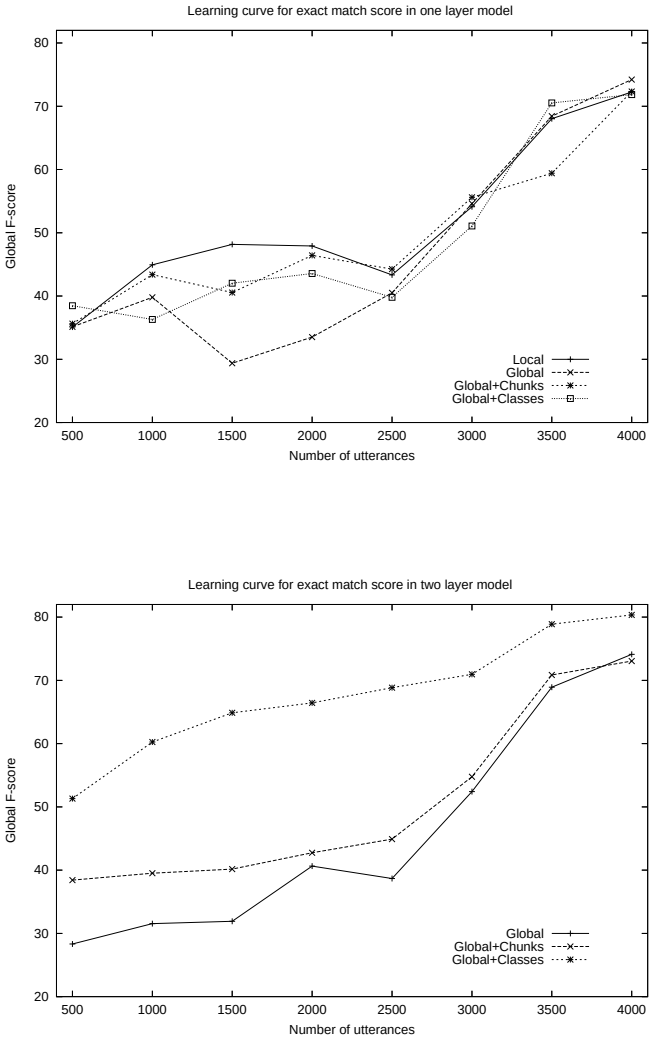


Figure 6.5: Learning curves for the *one layer* and *two layer* models. *Local* does not include links between hidden predicates. *Global* includes first Markov assumption. *Global + chunk* adds chunk information. *Global + classes* adds lexical classes.

Chapter 7

ML and Semantic Role Labelling

In this chapter we investigate the adequacy of Markov Logic [ML Richardson and Domingos, 2006] to perform Semantic Role Labelling (SRL). The goal of this chapter is to investigate the performance of the ML framework when using semantic representations which are more complex than frame-based, but which are shallow enough for dialogue system development. Previous work has shown the adequacy of using the SRL as part of the Spoken Language Understanding (SLU) module in a dialogue system [Tur et al., 2005]. In this chapter, we focus on the SRL task and leave aside the dialogue systems component. For this we use the PropBank [Palmer et al., 2005] and NomBank [Meyers et al., 2004] corpora. We use the CoNLL-08 shared task version of this corpora[Surdeanu et al., 2008]. We also adapted the CoNLL-05 version of the corpora to be compatible with the CoNLL-08. This will allow us to explore the use of different ML models in performing SRL. The results here presented here have been published in Riedel and Meza-Ruiz [2008] and Meza-Ruiz and Riedel [2009].

The chapter outline is as follows. Section 7.1 introduces the SRL task and presents our main approach to it. Section 7.2 presents the main elements of our ML model. Section 7.3 presents the experiments performed with the ML framework. Sections 7.4 and 7.5 present the results we obtained for those experiments. Finally, in 7.4.1 and 7.5.1 we discuss our findings.

7.1 Introduction

The Semantic Role Labelling (SRL) task aims to identify the semantic roles of the arguments of the predicates of a sentence (see subsection 1.2.3). Consider the following sentence and its semantic roles:

$$[_{ARG0}Ms.\ Haag]\ [_{play.02}\ plays]\ [_{ARG1}\ Elianti]$$

In this case, the predicate *play* has two arguments: the phrases *Ms. Haag* and *Elianti*. The semantic role of *Ms. Haags* is *ARG*0, which signals that this argument represents the actor who plays a role. The semantic role of *Elianti* is *ARG*1, which signals the role played by an actor. Table 7.1 summarises the semantic roles.

$ARG0-5$	Obligatory predicament arguments
AA	Argument adjunct
$AM-$	Prefix for argument modifier
$R-$	Prefix for discontinuous arguments
$C-$	Prefix for continuations
DIR	Directionals
LOC	Locatives
MNR	Manner marker
TMP	Temporal marker
EXT	Extend marker
REC	Reciprocal
PRD	Marker of secondary predication
PNC	Purpose clauses
CAU	Cause clause
DIS	Discourse Marker
ADV	Adverbial
MOD	Modal
NEG	Negation
STR	Stranded

Table 7.1: Table of semantic roles.

Traditionally the SRL task is divided into two stages [Carreras and Márquez, 2005]:

Argument identification: This identifies the arguments for a given predicate. Taking our example, it identifies the arguments for the given predicate, *play*. These arguments are the phrases *Ms. Haag* and *Elianti*.

Argument classification/labelling: It labels the predicate-argument pairs of the argument identification stage with their semantic roles. For example, in the case of the phrases associated with the *play* predicate, it assigns the role *ARG*0 to the argument *Ms. Haag*, and the role *ARG*1 to the argument *Elianti*.

Recent approaches to the task add two new stages to the task [Surdeanu et al., 2008]:

Predicate identification: In the previous cases, we assumed to know the predicate. In this new setup, the predicates of a sentence are unknown. For our example, this stage attempts to identify if the word *plays* represents a predicate.

Sense disambiguation: Since there are different frames for a predicate, which correspond to its different senses, the sense disambiguation stage aims to identify the sense of the predicate. The sense is represented by an identifier. In our example, this consists of signalling that the predicate *play* corresponds to its meaning represented by the identifier *02*. This sense corresponds to the meaning of playing a role, in contrast with the sense which denotes playing a game, represented by the identifier *01*.

Traditionally, the SRL systems model the task with a pipeline architecture. In this architecture, the output of one stage becomes the input of the next one. There are two predominant orders for the architecture: predicate identification, sense disambiguation, argument identification and argument classification [Johansson and Nugues, 2008], or predicate identification, argument identification, argument classification and sense disambiguation [Che et al., 2008]. The pipeline architecture has some disadvantages, one being that the early stages of the pipeline 'know' little about the latter stages of it. This results in error propagation. In order to minimise this, each stage can be fed with the predicted output of the previous stage in a cross-training setting, which increases the complexity of the training setting.

On the other hand, our system follows a joint approach in the spirit of Toutanova et al. [2005], in performing the above steps collectively. By using ML we incorporate the dependencies between the decisions of different stages in the pipeline and the well-known global correlations between the arguments of a predicate in a single model [Punyakanok et al., 2005]. Additionally, we jointly label the arguments of *all* predicates of a sentence.

Traditionally, the span of an argument was decided in term of constituent parses [Carreras and Márquez, 2005]. However, a recent reformulation of the task makes use of dependency parses [Surdeanu et al., 2008], and this is what we also do. In this case, we rely on the fact that the corresponding sentence is associated with a dependency tree. In this setup, the spans of the semantic roles are defined by the dependency tree. Figure 7.1 presents the dependency tree for our example sentence. Using this syntactic representation, we map the semantic arguments of the predicate *play.02* to a single

token. In this case, the word *Haag* is the head of the phrase *Ms. Haag*, and *Elianti* is head of the phrase *Elianti*. The semantic roles can be attached to these tokens and we obtain a equivalent representation, but one which is based on dependency trees. Figure 7.2 shows this representation.

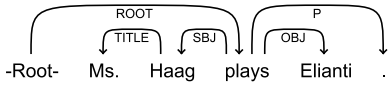


Figure 7.1: Dependency tree for *Ms. Haags plays Elianti.* sentence

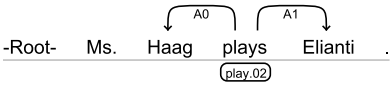


Figure 7.2: Semantic Role Labelling based on dependency trees for *Ms. Haags plays Elianti.* sentence

7.2 Model

In order to model the SRL task in the ML framework, we propose five hiddenpredicates consider the example of the previous section:

- *isPredicate/1* indicates that a word in a specific position is an SRL predicate. For instance in our example, *isPredicate(3)* signals that the lemma of the word in the position 3 is an SRL predicate (i.e., *play*).
- *isArgument/1* indicates the phrase for which its head is a specific position is an SRL argument. In this case, the predicate to which they are linked is undefined. For instance in our example, *isArgument(2)* signals that the phrase for which the word in position 2 is the head is an argument (i.e., *Ms. Haag*). In a similar way, *isArgument(4)* relates to the phrase *Elianti*.
- *hasRole/2* relates a predicate to an argument. For instance in our example, *hasRole(3,2)* and *hasRole(3,4)* relate the predicate in position 3 (i.e., *play*) to its two arguments.

- *role/3*, besides to relating predicate and argument, also relates this pair to a role. For instance in our example, *role(3,2,ARG0)* and *role(3,4,ARG1)* describe the roles that the predicate-argument pairs have. The phrase *Ms. Haag* refers to who plays a role which is signalled by *ARG0*, while *Elianti* is the role played which is signalled by *ARG1*.

- *sense/2* signals the sense identifier that a predicate in a specific position has. For instance in our example, *sense(3,02)* signals that the predicate in position 3 has the sense *02*. This indicates that the predicate *play* has to be read as playing a role, rather than playing a game.

These predicates are analogous to the four stages that are traditionally used in the SRL task. Figure 7.3 illustrates these predicates and the stages they belong to. However, the model includes an extra predicate – *isArgument/1* – which could be thought of as part of the argument identification stage. Other approaches avoid the decision process that *isArgument/1* implies, and decide directly if a word is an argument of a given predicate. We will investigate the consequences of including *isArgument/1* as a hidden predicate.

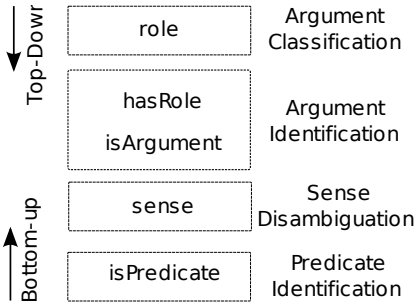


Figure 7.3: Architecture of the Markov Logic model compared with the traditional stages of SRL.

In addition to the hidden predicates, we define the following observed predicates:

- *word/2* orthography of a word for a given position.

- *lemma/2* lemma of word for a given position.

- *ppos/2* predicted POS tag of a word for a given position.

- *cpos/2* coarse POS tag of a word for a given position.

- *slemma/2* split lemma of a word for a given position.
- *sppos/2* split POS tag of a word for a given position.
- *dep/3* syntactic dependency and the head of a word for a given position.
- *link/2* syntactic head of a word for a given position.
- *frame/2* syntactic dependency frame of a word for a given position, includes direction and syntactic frames.
- *unlabelledFrame/2* syntactic dependency directions frame of a word for a given position.
- *path* syntactic dependency path between two words with different positions.
- *unlabelledPath* syntactic dependency directions path between two words with different positions.
- *pathframeDistance* distance of the syntactic dependency path between two words.
- *possiblePredicate/1* true if a word can be a predicate based on its POS tag.
- *possibleArgument/1* true if a word can be a argument based on its POS tag.
- *palmer/2* true if a potential argument can be an argument of a potential predicate (based in Xue and Palmer [2004]'s heuristics).
- *pruned/2* true if a potential predicate and argument should be pruned by Xue and Palmer [2004]'s heuristics.

7.2.1 Local formulae

The formulae are based on the work by Xue and Palmer [2004]. We define a local formula for each hidden predicate. The formulae for *isPredicate* and *isArgument* has the goal of capturing the immediate lexical and syntactic context of the word in order to determine if it is either an SRL predicate or argument. The formulae for *sense/2* tries to determine the sense of the predicate based on the most common sense for that predicate. This is done by a rule which relates the sense with the lemma of the predicate. In this formula the pair of lemma-sense which has the higher weights (most common) is assigned as the sense for such predicate. The formulae for *hasRole/2* looks

at the lexical and syntactic properties of a potential predicate and argument pair and tries to determine if they are related. The formulae for *role/3* captures the lexical and syntactic context of potential predicates and arguments pairs so we can determine the semantic role of each pair. The following are examples of the defined formulae (see Appendix C for the full list) :

$$\begin{aligned}
\phi_{ppos} &= lemma(i, l+) \wedge ppos(i, p+) \wedge isPredicate(i) \\
\phi_{children} &= lemma(i, l_i+) \wedge dep(j, i, _) \wedge lemma(j, l_j+) \wedge isArgument(i) \\
\phi_{deps} &= dep(_, i, l_i+) \wedge dep(_, j, l_j+) \wedge hasRole(i, j) \\
\phi_{lemmas} &= lemma(i, l_i+) \wedge lemma(j, l_j+) \wedge rolel(i, j, r+) \\
\phi_{lemma} &= lemma(i, l_i+) \wedge sense(i, s+)
\end{aligned}$$

The ϕ_{word} is a formula for *isPredicate/1* and it relates the POS tag and the lemma of a predicate. The $\phi_{children}$ collects all the lemmas of the syntactic children of an argument. The ϕ_{deps} relates the syntactic dependencies of a predicate to its argument. The ϕ_{lemmas} relates the lemmas of a predicate and an argument to their semantic role. Finally, ϕ_{lemma} relates the lemma of a predicate with its sense.

7.2.2 Structural constraints

Similarly to in the two layers model for semantic processing it is necessary to ensure that the hidden predicates are consistent with themselves (see 6.2.1.2). For instance, in order to ensure that the ground predicate *hasRole(3,2)* is true, we require *isPredicate(3)* and *isArgument(2)* to be true as well. It is easy to incorporate these constraints into the ML model. Table 7.2 shows the formulae for these constraints which we called *structural*. We divide it into two different groups, the *bottom-up* and the *top-down*. All these formulae are composed by implications. The *bottom-up* group of formulae consists of formulae where the antecedent of the implication is an SRL predicate of a latter stage in the pipeline architecture than the hidden SRL predicate of the consequent. In the *top-down* group we have the opposite case. For the first case, we have that the formulae are true every time an early stage SRL predicate is true or both SRL predicates are false or true at the same time. The effect of this is that every time an early stage SRL predicate is true, it forces the later stage SRL predicate to be true, like in a bottom-up pipeline. On the other hand, the top-down formulae does the contrary, every time the higher stage SRL predicate is true, it forces the lower stage SRL predicate to

be true, as in a top-down architecture. The effect of these two groups of formulae is tested during experimentation.

Bottom-up	$sense(p,s) \Rightarrow predicate(p)$ $hasRole(p,a) \Rightarrow predicate(p)$ $hasRole(p,a) \Rightarrow argument(a)$ $role(p,a,r) \Rightarrow hasRole(p,a)$
Top-Down	$isPredicate(p) \Rightarrow \exists s.sense(p,s)$ $isPredicate(p) \Rightarrow \exists a.hasRole(p,a)$ $isArgument(a) \Rightarrow \exists p.hasRole(p,a)$ $hasLabel(p,a) \Rightarrow \exists r.role(p,a,r)$

Table 7.2: Structural constraints for the ML model for the SRL task. These constraints ensure the SRL predicates predicted by the ML are well formed. *Bottom-up* simulates a bottom-up pipeline, where early stage predicates influence a later stage predicates. *Top-down* simulates a top-down pipeline.

7.2.3 Global formulae

Table 7.3 shows the global formulae of this model. The first two formulae, grouped under *unique labels*, constrain the number of *sense/2* and *role/3* predicates for a position p or the pair of positions p and a. Using the first formula, we force the model to only predict one sense for each predicate. In a similar manner, by using the second formula we force the model to only predict one role for each pair of predicate-arguments.

In the case of the *linguistically motivated* group of formulae, the first formula ensures that the role of a SRL predicate has only one proper argument. This forces the SRL predicate to only have one of the roles which start with *ARG*, but it can still have several modifier roles (e.g., temporal or adverbial arguments). Until now, the previous formulae were hard constraints. This means, they had to be satisfied by the solution found by the ML framework. However, the second formula of the linguistically motivated group is a soft global formula. This means it has a weight associated with it, similarly to the first and second Markov assumption formulae shown in the previous chapters (see section 5.3.2). This formula relates the sense of the predicate to the lemma of the predicate, and the POS tag of the argument. With this we intend to capture the influence that arguments have on the sense of the SRL predicate..

Unique labels	$sense(p,s_1) \wedge s_1 \neq s_2 \Rightarrow \neg sense(p,s_2)$ $role(p,a,r_1) \wedge r_1 \neq r_2 \Rightarrow \neg role(p,a,r_2)$
Linguistically motivated	$role(p,a_1,r) \wedge \neg mod(r) \wedge a_1 \neq a_2 \Rightarrow \neg role(p,a_2,r)$ $lemma(p,+l) \wedge ppos(a,+p) \wedge hasRole(p,a) \Rightarrow sense(p,+s)$

Table 7.3: Global formulae for ML model for the SRL task.

7.3 Experiments

There were two sets of experiments. The first type of experiments were based on the setting proposed by the CoNLL-08 shared task [Surdeanu et al., 2008]. These aim to compare the ML to current state-of-the-art systems, and while developing this system we analysed the effect of the structural formulae on the performance. These experiments were conducted using the PropBank and NomBank corpora. In the second set of experiments, we focused on improving the SRL system created with the first set of experiments, and on comparing the resulting system with a pipeline approach. However, in this second system we only focused on verbal predicates which corresponds to the setting proposed by the CoNLL-05 shared task [Carreras and Márquez, 2005]. We adapted this setting to be directly comparable to CoNLL-08 by using dependency parses rather than constituent parses.

7.3.1 Structural experiments

In these experiments we tested the effect of the structural constraints and global constraints on the task performance. For this we performed five experiments with our SRL system. The first experiment used our full system, which included the whole set of formulae previously explained. However, after analysing the results we were curious about the effect of the different types of formulae we included, and for this reason we performed four extra experiments where we omitted some of the formulae. This is the list of experiments performed:

Full: In this experiment we included all the formulae described in section 7.2.

Top-down: In this experiment we omitted the bottom-up formulae. As a result, the model behaves as a top-down system, but it is still a joint model.

Bottom-up: In this experiment we omitted the top-down formulae. As a result, the model behaves as a bottom-up system, but it is still a joint model.

Isolated: In this experiment we omitted both the bottom-up and top-down formulae. This resulted in a joint model which does not ensure that the SRL predicates are well formed.

Structural: In this experiment, we omitted the linguistically motivated formulae.

For these experiments we used the Open track CoNLL08 shared task setup Surdeanu et al. [2008]. In the open track, the SRL system has access to extra resources such as named entities, WordNet information, and predicted syntactic dependencies. This contrasts with the Closed track setup, in which systems have to perform dependency parsing and the SRL task in a joint manner. Since our interest is focused on the SRL task, we chose the Open track in which we do not need to perform syntactic parsing.

Our models were created using the standard setup for ML software [Riedel, 2007]. In this setup we have to identify the number of iterations which are identified using the development corpus. The maximum number of iterations for the MIRA algorithm was 5.

7.3.2 Joint vs pipeline experiments

For these experiments we only use the PropBank [Palmer et al., 2005] version of the corpus. The setting of the task is the same as that defined by the CoNLL-08 Shared task [Surdeanu et al., 2008]. In this set of experiments, we aim to compare our SRL system with a pipeline system. We also look to improve the performance of the system from the previous experiments. We do this by creating a new model and cleaning some of the rules from the previous model. The main modifications are:

1. We change the source of dependency parses. For this system we use the parses used in the CoNLL-05 Shared task. Originally, the CoNLL-05 parses were based on the constituents, but we converted them into dependency parses by using the software provided by [Johansson and Nugues, 2007].

2. Unification of the use of *possiblePredicate/1* and *possibleArgument/1* predicates in the formulae of the model. Originally, these predicates only appeared in local formulae for the *hasRole/2* and *role/3* hidden predicates. We modify the formulae to include these predicates for all hidden predicates. These predicates have the purpose of pruning the potential predicates and arguments.

3. We increase the local formulae for *sense/2* hidden predicates.

4. We fix some minor errors in some of the original formulae.

With the new model we investigate the effect of the joint learning provided by the ML framework. In particular, we investigate the effect of adding to the model *isArgument/1*. Recall that this predicate is part of the argument identification stage together with the *hasRole/2* predicate (see Figure 7.3). Traditionally, approaches do not try to identify if a word is potentially an argument regardless of its predicate. They decide if a word is argument depending on the already identified predicate [Johansson and Nugues, 2008; Che et al., 2008]. Additionally, we would like to compare the joint model with a pipeline approach. To this end, we perform the following experiments.

Full: In this experiment we used a Markov Logic Network with all local and global formulae described in the previous section.

Bottom-up: In this experiment we removed the structural top-down constraints from the complete model, in a similar way to in the structural experiments.

Full-arg not argument: In this experiment, we removed the *isArgument/1* hidden predicate, from the *Full* system.

Bottom-up not argument: In this experiment, we remove the *isArgument/1* hidden predicate from the *Bottom-up* system.

Pipeline In this experiment, we split the task into three stages: predicate identification, as one stage, argument identification and sense disambiguation, in the following stage, and finally role identification as an independent stage. We implement these stages using the ML framework and the same set of rules.

Pipeline bottom-up: In this experiment we used the Pipeline setting but for the argument identification and sense disambiguation stage we remove the top-down structural constraints.

Pipeline bottom-up not argument: In this experiment we used the Pipeline setting but for the argument identification and sense disambiguation stage we removed the top-down structural constraints and the *isArgument/1* hidden predicate.

The two first models set the baseline for our results. If the joint model is helping the task, these two experiments should give better performance than the following

experiments. The *no arg* experiments will allow us to check how good our model is without the *isArgument/1* hidden predicate. The results of these experiments will help us to determine if identifying the arguments jointly helps the task. Finally, by doing the pipeline experiment we will be able to decide if the joint strategy provided by the ML framework helps the performance of the SRL task.

7.4 Results of the structural experiments

This section presents the results of the experiments on the ML model for Semantic Role Labelling (SRL) in the context of the Open Track of the CoNLL 2008 Shared Task [Surdeanu et al., 2008]. We achieved a semantic F-score of 74.59%, the second best in the Open Track, which points to the adequacy of the ML framework for performing SRL [Riedel and Meza-Ruiz, 2008]. Further experimentation allowed us to improve this result for the in-domain testing corpus (i.e., WSJ corpus).

Table 7.4 summarises the F-scores for the *in-domain* (i.e., WSJ corpus) and *out-domain* (i.e., Brown corpus) test using the evaluation for the CoNLL-08 shared task [Surdeanu et al., 2008]. Interestingly, the best model for the in-domain evaluation is the bottom up (result labelled as Bottom-up). In section 7.4.1 we analyse why this could be. The best result for the out-domain evaluation is the full model (result labelled as Full). The top-down does not perform as well as the two first models, which was an expected result, since it is harder to come up with a *role/3* predicate which is correct and forces predicates like *hasRole/2* to be right (result labelled as Top-down). When we eliminate both bottom-up and top-down constraints the performance drops drastically (result labelled as Isolated). This was expected since there is nothing which enforces the well formed SRL predicates. When we omit the global formula (result labelled as Structural), we notice that the performance slightly drops. This shows the effect of using global features in the model.

For comparison, Table 7.4 also shows the performance of the best system in Closed track and Open Track of the CoNLL 2008 Shared Task [Vickrey and Koller, 2008; Johansson and Nugues, 2008]. Our results are directly comparable with the Closed Track result which uses the same experimental setup as ours. Our result is not directly comparable with the Open Track result, however it gives a good reference of what to expect from our system in the future. We found that some of the difference between the performances of the Open and the Closed systems comes from the different quality of the dependency parses. The parses of the Closed track are of lower quality, as is

Model	Development	In-domain (WSJ)	Out-domain (Brown)
Full	74.32%	75.72%	**65.38**%
Bottom-up	75.15%	**76.96**%	63.86%
Top-down	69.65%	73.48%	59.34%
Isolated	56.48%	60.49%	48.12%
Structural	72.17%	74.93%	64.23%
Best-system Closed-track	N/A	81.75%	69.06%
Best-system Open-track	N/A	76.17%	66.23%

Table 7.4: CoNLL08 F1 score performance for different variations of the ML model from evaluation. To compare we include the best results of the Closed and Open tack of the CoNLL-08 shared task.

Model	In-domain (WSJ)	Out-domain (Brown)
Best dependency parser in Open track	90.13%	82.81%
Dependency parser in Closed track	85.50%	77.06%

Table 7.5: Labeled Attachment Score of the syntactic dependencies of the best system on the Closed track and the parses used on the Open Track.

shown in Table 7.5.

An important result for our system was the exact match evaluation for the out-domain test corpus. The scores are presented in Table 7.6: between parentheses we include the position of the systems in relation to the other ones (including open and closed track systems). In the case of in-domain evaluation we are in 6th position out of 24 systems. However, in the case of out-domain evaluation we are obtained the 2nd highest score. We hypothesise that the joint modelling helped the model to perform better. However, further research in this direction is required.

An interesting question arises about the practical performance of our SRL system. To analyse this aspect, we collected the training and testing times for our experiments. Table 7.7[1] shows these times measurements. We observe that the top-down structural constraints tend to make the inference process slower. This is related to the fact that such formulae have existential quantifiers which makes access to the ground predicates more complex. We notice that with the bottom-up model we obtain the best trade-off

[1] These times were obtained using an Intel(R) Xeon(TM) CPU 3.80GHz computer with 1GB of RAM memory.

Model	In-domain (WSJ)	Out-domain (Brown)
Full	42.77% (6)	31.15% (2)
Best-system Close-track	56.12% (1)	36.90% (1)
Best-system Open-track	46.94% (5)	30.28% (4)

Table 7.6: CoNLL08 exact match score from evaluation performance for different models.

Model	Train Time	Test Time
Full	25h	24m
Bottom-up	11h	14m
Top-down	22h	23m
Isolated	11h	14m
Structural	22h	33m

Table 7.7: Time performance for different models.

in terms of speed and performance.

7.4.1 Error Analysis

The results presented in the previous section showed that the Full model does not perform as well as the Bottom-up for the in-domain test. This was not expected since the Full model was designed to find ML predicates which are consistent with the final SRL predicates. For instance, when a *hasRole(p,a)* predicate is true, the full model should find a configuration of ground predicates where *isPredicate(p)* and *isArgument(a)* are true (note the positions *a* and *b* have to be the same for the between *hasRole/2* and *isPredicate/1* and *isArgument/1* predicates). However, the inclusion of the top-down constraints did not help the full model on the contrary, they made it worse. Analysing the errors in the full model for the development set we found the following 5 false positives to be most typical:

- `AM-TMP` (190)
- `A2` (303)

- `R-AA` (351)
- `A0` (504)
- `A1` (815)

The numbers between parentheses represent the amount of times this label was wrongly proposed. From this list we notice the label `R-AA` which in the training corpus had a low frequency (it only appeared once), however in the Full model labelling occurred 351 times. In the case of the other labels, it is expected to be in the most common errors since these are the most frequent role labels.

We found that these errors were an artefact of the training. The top-down formulae makes sure that the *role/3* predicate of the solution is related to the right *hasRole/2* predicate. Therefore, in order to force a *role/3* predicate to be true, this predicate relies on *hasRole/2* being true. Therefore, the weights related to *role/3* only need to ensure that the right role label wins. When analysing the case of the `R-AA` role, the weights are modified to choose this role, but other roles have weights in the same scale. However, in the bottom-up case, which does not contain the top-down formulae, the link between *role/3* and *hasRole/2* is not present. In this case, the *role/3* predicate weights have to ensure that the cumulative weights among the roles are non negatives, otherwise it would not assign a role for that position. In this case, the common roles end up with larger weights than non common roles. This is the reason why, when we analysed the errors within the Bottom-up development set, we do not find the `R-AA` label, as shown in the following list:

- `AM-LOC` (93)
- `AM-TMP` (163)
- `A2` (206)
- `A0` (436)
- `A1` (717)

The differences between the systems can be appreciated in Figure 7.4. This shows the semantic roles for the sentence *Whatever its merits, Sony's aggressive defense is debilitating for Justin.*. The SRL labelling is presented as a graph, where the words are the heads of phrases and the labelled links relate predicates to their arguments.

We found that all systems identified the position of the word *debilitating* as predicate. In the gold standard labelling *debilitating* was not labelled as a predicate, although it should have been. We believe that this was due to a mislabelling in the POS tag of this word (i.e., the adjective tag *JJ* instead of a gerund tag *VBG*). The systems were able to deal with this situation. However, the satisfaction of the structural and global formulae led to different errors. In the case of the Full system, we see that this predicate produced the extra arguments (i.e., $A1$ and $A2$ with dashed lines). More interesting is that this example shows the role `R-AA` for the predicate *defense*. We identify this error as being an artefact of the training and the top-down structural constraints which are contained in this model. However, this error does not exist in the bottom-up model, as it does not propose `R-AA` as role. But the lack of this constraint results in not proposing the role $A0$ for predicate *merit*. Exploring the resulting ground ML for this example we found that the *hasRole/2* predicate was true, but because we lack of the top-down constraints, the *role/3* predicate for this network was not enforced, and it is missing from this labelling.

The Top-down and Structural systems both labelled the `R-AA` role again. This is because the top-down structural constraints are present in both systems. Notice that the Structural system added the role $A2$ for *merits*. This is because the global formulae is not present for this system, and there is no way to tell that this is an unlikely argument for the predicate *merit*. The Isolated system fails to identify all the senses of the predicates. This is because there is no link between the *isPredicate/1* and the *sense/2* predicates. Therefore there is nothing which forces it to assign a sense to these SRL predicates.

When we analyse the difference between verbal and nominal SRL predicates we find that the latter are harder to label. Table 7.8 shows the results for the verbal and nominal predicates. Notice that the performances are lower for the case of the nominal. The difference in performance between the Open and Closed Track systems is reduced when we only evaluate nominal predicates. In the case of the verbal predicates, it is of 7% for the in-domain test (between the best systems). However, in the case of nominals this difference is only of 2%. In the case of the out-domain corpus, our system has the best performance for the nominal predicates.

This difference was expected considering that there is a significant body of previous work that has analysed the SRL problem on PropBank, but minimal work on NomBank. This is the case for our model, whose formulae is based on previous work in Propank [Xue and Palmer, 2004]. In particular, the nominal predicates involve fewer

Model	In-domain (WSJ)	Out-domain (Brown)
Verbal predicates		
Full	78.72%	66.57%
Best-system Close-track	86.37%	71.87%
Best-system Open-track	79.75%	69.57%
Nominal predicates		
Full	71.03%	60.17%
Best-system Close-track	75.42%	60.13%
Best-system Open-track	73.29%	53.25%

Table 7.8: CoNLL08 exact match score from evaluation performance for different models.

syntactic dependencies than the verbal. This is because they involve only noun phrases. This puts much more demand on the lexical features of the models. At this point, it is difficult to come up with a final explanation as to why it is missing in the nominal predicates. Further research is necessary to this end.

7.5 Results of joint vs pipeline experiments

This section presents further experimentation with the SRL system presented in the previous section. From the previous experiments, we were able to identify the advantages of the Bottom-up model and why the Full model performed worst. With the pipeline experiments, we look into the joint aspect of our model. Particularly, we investigate the difference between a pipeline SRL system versus our joint model. We also look into two parts of our system the use of the *isArgument/1* predicate which captures the potential of a token of being an argument despite the SRL predicate. Table 7.9 presents the main results in this direction.

Table 7.9 shows the F-scores results of the joint vs pipeline experiments. There are three main points to notice:

1. We repeated our experience with the structural experiments and we found that the Bottom-up model provides a better performance than the full model. We noticed the main factor is the same as that explained in the error analysis of the structural experiments (see 7.4.1). This is an artefact of the training process. On

Model	In-domain (WSJ)	Out-domain (Brown)
Full original model	78.72%	66.57%
Best-system [Johansson and Nugues, 2008]	86.37%	71.87%
Full	79.09%	67.64%
Bottom-up	**80.19**%	**68.02**%
Full *noArg*	78.17%	66.84%
Bottom-up *noArg*	79.37%	66.70%
Pipeline	78.19%	64.66%
Pipeline bottom-up	79.28%	65.24%
Pipeline bottom-up *noArg*	78.99%	64.70%

Table 7.9: F1-scores for the PropBank ML model for full model and pipeline version. Full model includes all the formulae. Bottom-up does not include the top-down structural formulae. *noArg* omits the *isArgument/1* hidden predicate. Finally, pipeline systems test the traditional pipeline architecture.

the other hand, we notice that our performance is still 6% lower than the current best system. Part of this is because the quality of the dependency parses. However, we noticed we have increased our performance by 1.47% compared with our previous system. The improvement was obtained despite using worse parsers than in the structural experiments (i.e., 72.02% Labelled Attachment Score) [2].

2. The results justified the use of the *isArgument/1* hidden predicate. In each of the experiments where we omit it, the performance is lower.

3. The results show that the pipeline approach produces slightly worse results than the joint model. The differences between approaches is larger when we pay attention to the time performance. Table 7.10 shows the times for the fastest SRL system. We notice that the times are longer for the pipeline system.

7.5.1 Error analysis

While analysing our development results for the Bottom-up system from the experiments above described, we notice the following types of errors:

[2]Given the restrictions on the availability of the CoNLL-08 Shared Task corpus, we adapted the CoNLL-05 Shared Task corpus to the CoNLL-08 version of the corpus. However, this adaptation required to use worst parses.

Model	Train time	Test time
Bottom-up noArg	3.84h	14.02
Pipeline bottom-up noArg	4.61h	26.44m

Table 7.10: Time performance for fastest Pipeline and Bottom-up models[3].

- The lack of top-down structural constraints between *isPredicate/1* and *sense/2* hidden predicates makes our system find predicates which have not been assigned a sense, this means they are not disambiguated. In order to fix that, the model has to include the structural constraints for these hidden predicates, or a post-processing step has to assign a typical value (e.g., "01"). We think the first option is the most appropriate.

- A main source of errors is prepositional phrases. These are introduced by the dependency parser. Seven of our top ten most common errors contain semantic roles which have as an argument a preposition, while the rest are punctuations, the verb "to be" and relative clauses. In particular, the problem In future work we will look into using better parsers.

Figure 7.5 presents the labelling for three of the models for the utterance: *–If Dow Industrials fall 25 points at opening, contract pauses for 10 minutes.*. With these labellings we can compare the Bottom-up model against the pipeline bottom (the best results for the pipeline system) and the bottom-up when not using the *isArgument/1* hidden predicate. The figure shows the effects of the joint model. The pipeline model decides that *pause* is not a SRL predicate, and since this decision is passed to the following stage, this can not be changed. In the case of the Bottom-up, we found that even though both model share similar local formulae, which would make the likelihood of the Bottom-up model change this situation, in this model it is decided that *pause* is an SRL predicate. The labelling for the Bottom-up without *isArgument/1* gives us more insight into what made this possible. We notice that this labelling is equal to the pipeline one. That is, *pause* is not a SRL predicate. We believe this is because *isArgument/1* found that *If* should be an argument, and it forces to *isPredicate/1* to find a predicate for it. However, when we omit this hidden predicate, the task of identifying argument relies on the *hasRole/2*, predicate which has more sparse weights and it does not find *If* to be an argument, and so it does not force the model to find a predicate for it.

7.6 Discussion

In this chapter we have shown a set of models for Semantic Role Labelling (SRL) which compares with state-of-the-art performances of this task. In this chapter we investigated the impact of structural constraints on the performance of the system. We found that our Full model was affected by a training artefact which makes it worse than the simpler version of the Bottom-up model. In particular, we found that the Full model could not distinguish among low frequency roles labels from high frequency ones. This analysis provided us with a new model which improved our original performance by 1.24% for in-domain testing. Additionally, we discovered that the Bottom-up model provided the best trade-off between performance and training times. The Bottom-up model used half the time during training. Further research with this model allowed us to increase the performance in the PropBank corpora by another 1.46%. This was achieved despite using worse dependency parses.

The results give us an insight into the interworking of the Markov Networks used by the ML framework. We found that the structural constraints ensured good performance. However, the type of structural constraint is important. Bottom-up types perform better; the Top-down type is not able to separate low frequency labels from high frequency ones. We found that such constraints are important to the model, since not including none at all results in a model which produces ill-formed SRL labellings (see results for Isolated model). We also noticed the positive impact of the global formulae in the model: without these formula the performance drops by 0.79%. We looked into the joint aspect of our model and tested the factor of predicting the potential of a word of being an argument regardless of its predicate. We found that this was an important addition to our system. Additionally, we tested our system against a pipeline system. We found that the joint model performed better and was faster than the pipeline system. These results point out the adequacy of the ML framework to perform the SRL task.

As shown by Tur et al. [2005] SRL is a promising semantic representation for performing semantic processing for dialogue systems. The shallow nature of the representation and the fact that it depends on off-the-shelf resources (e.g., POS taggers, lemmatisers, and constituent/dependency parses) make it a good candidate for the development of dialogue systems. With the results obtained in this chapter and presented in section 7.4 and section 7.5, we have shown the adequacy of the Markov Logic framework for handling a more complex semantic representation than a frame-based

representation. This is a new direction for semantic processing, and further experimentation should be done done on the use of our SRL system in dialogue systems, particularly with scenarios of limited resources.

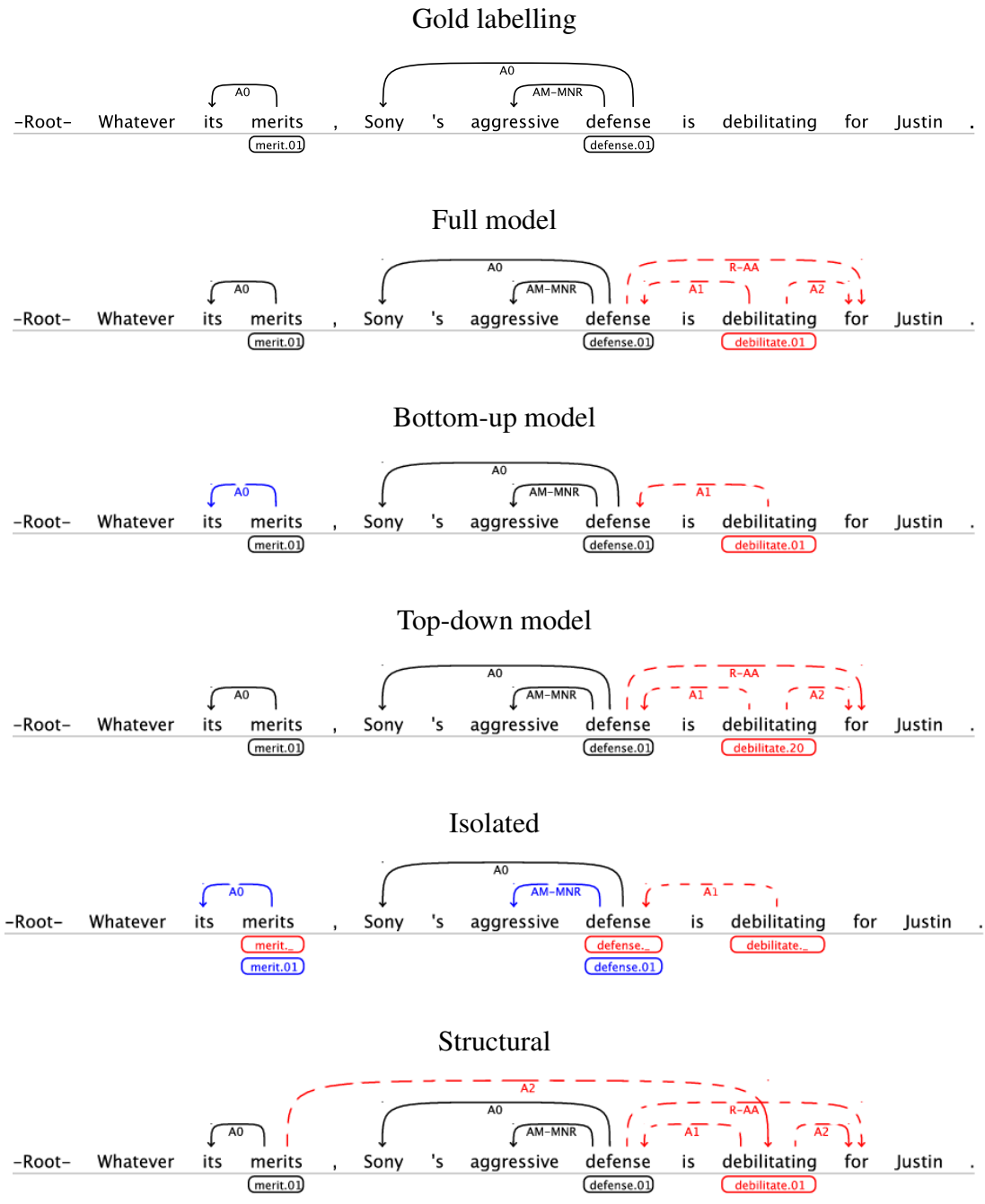
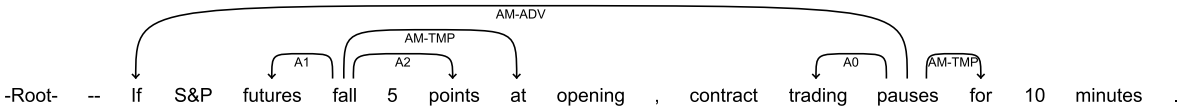


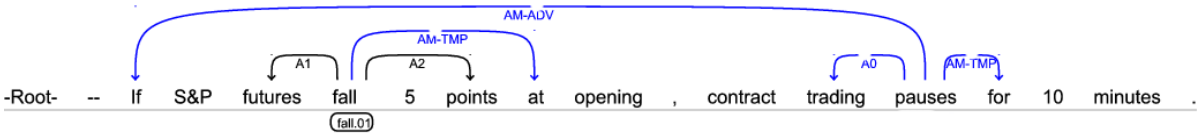
Figure 7.4: SRL Labellings produced for each one of the ML model presented in this section for the sentence *Whatever its merits, Sony's aggressive defense is debilitating for Justin.* as a graph. Solid and/or black lines are true positives; dashed and/or red lines are false negatives; and broken and/or blue lines are false positives.

Gold labelling



Bottom-up model

Bottom-up no arg



Pipeline bottom-up

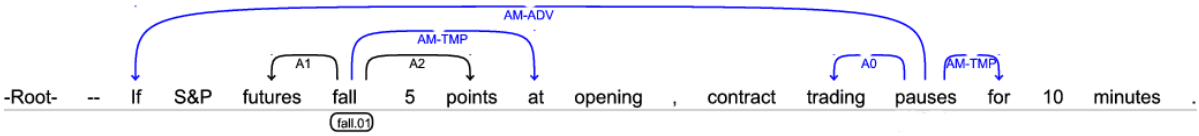


Figure 7.5: SRL Labellings produced for some of the systems presented in this section for the sentence *–If Dow Industrials fall 25 points at opening, contract trading pauses for 10 minutes.* as a graph. Solid and/or black lines are true positives; dashed and/or red lines are false negatives; and broken and/or blue lines are false positives.

Chapter 8

MLN in a working dialogue system: TownInfo

In this chapter we present a Markov Logic [ML, Richardson and Domingos, 2006] semantic processing module for the TownInfo dialogue system [Lemon et al., 2006a]. The development of this module is a real world test case scenario; in this case, a semantic processing module for the TownInfo system was required. There was available a corpus of dialogues between human users and the dialogue system, which was labelled directly with a shallow semantic representation that the dialogue manager could recognise (see section 3.3). The resulting corpus met our requirements for this work: it had limited resources (none in fact), it was relatively small, containing approximately 2000 utterances, and it had a system-dependent semantic representation which had not been modelled before. In this chapter we present the ML model used for the semantic processing task. In order to test the adequacy of the model we perform some experiments under two different scenarios: with manual transcriptions, and with automatic transcriptions.

The outline of the chapter is as follows. Section 8.1 presents those properties of the TownInfo semantic representation which are particularly challenging. Section 8.2 presents the ML model used for the semantic processing task. Section 8.3 presents the experiments that were performed on the ML model. Section 8.4 presents the results of those experiments. Finally, section 8.5 presents a discussion and summary of our findings.

8.1 TownInfo semantic representation

The TownInfo corpus uses as a semantic representation a set of triples which encode the speech act and the slot values of an utterance. Figure 8.1 reproduces the examples of the corpus introduced in section 3.3. As described in this section, the semantic representation in the TownInfo is differs from the frame-based semantic representation used in ATIS corpus in three aspects:

1. Each slot-value pair has a speech act related to it. For instance, the first utterance of our examples contains triplets in which each first element is a speech act.

2. Values are not necessarily a sequence of words contained within the utterance. For instance, in the second utterance of our examples, the value is `null`, which is not present in the utterance.

3. The corpus contains instances of utterances without meaning. For instance, in the third utterance of our examples, the utterance did not have any contribution to the dialogue.

These differences make it impossible to use any of the ATIS/Communicator models (see sections 5.3 and 6.2.1). However, it is possible to design a new model within the ML framework which would be tailored to this semantic representation. This is a similar case to that presented for the Semantic Role Labelling model in Chapter 7; in that chapter we created a ML model specific to the Semantic Role Labelling task (SRL).

Before presenting the model for TownInfo, I will present some insights into the design and development of a new model in the ML framework. The designing of a ML model consists of four stages:

- Identify types and constants of the semantic representation: these are the elements which constitute the semantic representation. For the TownInfo corpus we identified: speech acts such as *provide information* and *wh question*; slots as *food type*, *task* and *phone number*; and values such *French* and *centre*. Notice that there are two type of values: the ones which are present in the utterance, like *French* and the ones which are not, like *null*.

- Define predicates. These are used to relate the elements of the semantic representation to the utterance. We do this by using the position of tokens in an utterance as an argument of the predicate. For instance, *sa/2* is the predicate for

---

looking for a French restaurant in the centre of the town

`(prov_info,food_type,french),`

`(prov_info,task,restaurant),`

`(prov_info,region,centre)`

could you repeat the phone number please

`(wh_question,phone_number,null)`

garden

`( )`

Figure 8.1: Example of semantic triplet labellings from the TownInfo corpus. The first utterance shows a set of triples, with each one having a word of the utterance as its value. The second utterance shows a triplet whose value is not part of the utterance (i.e., `null`). The third utterance shows an empty triplet: this means there is no valid semantic representation for this utterance.

speech acts: the first argument is the position of the token, while the second argument is the speech act. For the semantic representation of TownInfo, we have identified the following predicates: one for speech acts, one for slots with values in the utterance, and one for slots for values which are not in the utterance. These predicates are hidden, and they model the semantic representation. In terms of observable predicates, we only use the orthography of the word, since this is the only information available.

- Identify the constraints that the predicates have to satisfy. For instance: there is only one speech act for a given value, or for each speech act there is a slot.

- Define the local and global formulae of the model. This step consists of proposing formulae which involve both models. A good approach is to start with local formulae. Once these produce an acceptable performance, it is a good idea to experiment with some global formulae, and evaluate if their use results in an improvement by using it. The local model defines a baseline which might be possible to improve by the use of global formulae.

The two last stages require the most time of the development process. The designer has to propose new formulae, and test them with a development set. If the result of the test are not satisfactory, the designer has to experiment new formulae which capture more

relation between the predicates. Most of the time these formulae are standard like the formulae for a slot and the orthography of the words in certain windows that we have explored in previous models. Global formulae are more difficult to develop, because most of the time these are intuitive (e.g., the relation among slot and any speech act), but our experience also shows that most of these do not necessarily increase performance. The following section presents the final model used for the semantic processing module for the TownInfo dialogue systems.

8.2 MLN model

The ML model for the TownInfo system has six hidden predicates. Consider the example from the previous section:

- *sa/2* relates a token in a specific position to a speech act. For instance, in the first example utterance, *sa/2* is *sa(4,provide_info)*, for the first triplet.
- *slot/3* relates a slot to the sequence of words which represent its value. For instance, for the first example utterance we have *slot(4,4,food_type)*, for the first triplet.
- *value/1* signals if a token represents a value for a slot, similarly, to the *bio* predicate in the improved model for the ATIS/Communicator corpus. However, in this case we do not distinguish between being inside or outside of the slot. For instance, in the first example utterance, *value(4)* signals that the token in position 4 (*French*), is part of a value.
- *triplet/3* is a predicate for a triplet whose value is not part of the utterance. For instance, for the second example utterance, we have the predicate *triplet(provide_info, phone_number, null)*.
- *hasTriplet/0* is true if the utterance has a triplet whose value is not in the utterance. That is, if one or more *triplet/3* predicates should present.
- *hasSlot/0* is true if the semantic representation has a slot whose value is part of the utterance. When *hasTriplet/0* and *hasSlot* are both false, this means that the semantic representation is an empty triplet, as is the case in the third example utterance.

Notice that the model for TownInfo is a multiple predicate model, where predicates are related to each other. The model is divided into two parts: slots with a value within the utterance and slots with a value that is not in the utterance. The *hasSlot/0*, *sa/2*, *slot/3* and *bio/2* predicates capture the slots which have as a value a word within the utterance. On the other hand, the *hasTriplet* and *triplet/3* predicates capture the case for when the value is not part of the utterance. In combination, these predicates allow a represent the semantic representation of the TownInfo corpus to be produced.

In addition, to the hidden predicates we also define the observed predicate:

- *word/2* orthography of a token for a given position.

This is the only observable predicate since there is no extra information available. However, it is easy to extend the model to support extra information as was done with the ATIS/Communicator module (see subsection 6.2.1).

8.2.1 Local formulae

The model contains local formulae for each of the hidden predicate. The formulae for *slot/3* and *value* predicates are triggered within a window of words in which the predicates can occur. The intuition behind this setup is that the local aspects of the utterance define the arguments of these predicates. In our example, *for*, *a*, *French* and *restaurant* are good indicators that the *French* is a value, and that this value is a food type.

On the other hand, the *sa/2*, *hasSem/0*, *triplets* and *hasTriplets* predicates depend upon the whole utterance, since they are not associated with local elements of the tokens where they appear. For instance, in the second utterance of our examples, the speech act of the triplet depends on the starting words *could you repeat*. Intuitively, this does not appear be the case for the *sa/2* predicate, since it is associated with a specific position in the utterance, however, the speech act depends on the other bits of the sentence. For instance, in the first utterance of the examples, *looking* has an influence on the type of the speech act. We observed this phenomenon in the ATIS/Communicator corpora, and we again confirmed its occurrence during the development experimentation within the TownInfo corpus.

These are the local formulae used in our final model :

$$\begin{aligned}
\phi_{slot/word_{ini}} &\quad word(i,+w) \wedge slot(i,_,+s) \\
\phi_{slot/word_{-1}} &\quad word(i-1,+w) \wedge slot(i,_,+s) \\
\phi_{slot/word_{-2}} &\quad word(i-2,+w) \wedge slot(i,_,+s) \\
\phi_{slot/word_{last}} &\quad word(f,+w) \wedge slot(_,f,+s) \\
\phi_{slot/word_{+1}} &\quad word(f+1,+w) \wedge slot(_,f,+s) \\
\phi_{slot/word_{+2}} &\quad word(f+2,+w) \wedge slot(_,f,+s)
\end{aligned}$$

$$\begin{aligned}
\phi_{value/word_{ini}} &\quad word(i,+w) \wedge value(i,+v) \\
\phi_{value/word_{-1}} &\quad word(i-1,+w) \wedge value(i,+v) \\
\phi_{value/word_{-2}} &\quad word(i-2,+w) \wedge value(i,+v) \\
\phi_{value/word_{+1}} &\quad word(i+1,+w) \wedge value(i,+v) \\
\phi_{value/word_{+2}} &\quad word(i+2,+w) \wedge value(i,+v)
\end{aligned}$$

$$\begin{aligned}
\phi_{sa/word} &\quad word(_,+w) \wedge sa(_,+sa) \\
\phi_{triplet/word} &\quad word(_,+w) \wedge triplet(+sa,+s,+v) \\
\phi_{hasTriplet/word} &\quad word(_,+w) \wedge hasTriplet() \\
\phi_{hasSlot/word} &\quad word(_,+w) \wedge hasSlot() \\
\phi_{hasSlot/bigram} &\quad word(i,+w_1) \wedge word(i+1,+w_2) \wedge hasSem()
\end{aligned}$$

As mentioned before, notice that *sa/2*, *triplet/3*, *hasTriplet* and *hasSlot* have similar local formulae (i.e., $\phi_{/word}$). These relate the predicates to any word in the utterance. The only difference is that *hasSlot* which adds a bigram formula which considers pairs of tokens in the utterances, rather than single tokens.

8.2.2 Hard constraints

These formulae attempt to ensure that the solution found by the ML framework is well-formed. To this end we add the following formulae:

$$\begin{aligned}
\phi_{1_{arg}} &\quad word(i,_) \wedge value(i,a_1) \wedge a_1 \neq a_2 \Rightarrow \neg value(i,a_2) \\
\phi_{hasTriplet} &\quad hasTriplet() \Rightarrow \exists a,s,v.triplet(a,s,v) \\
\phi_{notHasTriplet} &\quad \neg hasTriplet() \Rightarrow \forall a,s,v.\neg triplet(a,s,v) \\
\phi_{hasSlot} &\quad hasSlot() \Rightarrow \forall i,j,s.\neg slot(i,j,s) \\
\phi_{noHasSlot} &\quad \neg hasSlot() \Rightarrow \exists i,j,s.slot(i,j,s) \\
\phi_{sa/slot} &\quad slot(i,j,s) \Rightarrow \exists a.sa(i,a) \\
\phi_{slot/sa} &\quad sa(i,a) \Rightarrow \exists j,s.slot(i,j,s)
\end{aligned}$$

Formula $\phi_{1_{arg}}$ ensures that there is only one *value/2* predicate per token. The $\phi_{hasFlag}$ and $\phi_{notHasFlag}$ determine whether or not there are *triplet/3* predicates present in the solution. The ϕ_{empty} and $\phi_{notEmpty}$ formulae have a similar goal, but with the *slot/3* predicate. Finally, $\phi_{sa-slot}$ and $\phi_{slot-sa}$ are structural formulae for the *sa/2* and *slot/3* predicates. They ensure that if a *slot/3* predicate is true for a position *i*, there is a *sa/2* predicate for the same position. These two formulae are similar to the structural formulae introduced for the SRL model (see section 7.2).

8.2.3 Global formulae

In order to create semantic representations of the TownInfo corpus we need to relate the hidden predicates to be related between them. For example, we need to ensure if the predicate *hasTriplet/1* is true then there is a predicate *triplet/3*. As in the previous model, the global formulae is constrained by positive or negative weights. Recall that negative weights punish bad solutions, and positive weights reward good solutions. We constrain the weights to these values because they facilitate to have a faster performance.

We start by defining the formulae for the *triplet/3* predicate. These formulae have negative weights. In this case, we want to ensure that anytime that *hasTriplet* is true, the model punishes the bad arguments of the *triplet/3* predicate . We do this with the

following formulae

$$
\begin{aligned}
-\phi_{hasTriplet/sa} \quad & word(\_,+w) \wedge hasTriplet() \wedge triplet(+sa,\_,\_) \\
-\phi_{hasTriplet/slot} \quad & word(\_,+w) \wedge hasTriplet() \wedge triplet(\_,+s,\_) \\
-\phi_{hasTriplet/value} \quad & word(\_,+w) \wedge hasTriplet() \wedge triplet(\_,\_,+v)
\end{aligned}
$$

These formulae are only triggered when *hasTriplet/0* is true. In these formulae, the weight is factored into each argument of the *triplet/3* predicate. The predicate has three arguments, and therefore there are three formulae: one for speech act, one for the slot and one for value arguments. The symbol + within the formulae indicates that the weights depend on these arguments and on any word in the utterance. For instance, in the second example utterance, the starting words *could you repeat* would subtract weight if the wrong speech act was proposed, and a similar situation would happen with the words *phone number*, which would expect to subtract weight if a slot different to *phone_number* was proposed. It is harder to illustrate the case for the *null* value, but the idea is similar, in that in the presence of a different value, the formula punishes the solution.

The following formulae relate the *slot* and *value* predicates.

$$
\begin{aligned}
-\phi_{ini} \quad & word(i,+w) \wedge slot(i,j,+s) \wedge value(i,+v) \\
-\phi_{ini*} \quad & word(i,\_) \wedge slot(i,j,+s) \wedge value(i,+v) \\
-\phi_{last} \quad & word(j,+w) \wedge slot(i,j,+s) \wedge value(j,+v) \\
-\phi_{last*} \quad & word(j,\_) \wedge slot(i,j,+s) \wedge value(j,+v)
\end{aligned}
$$

The first formula relates a word in the position i to its value and to the slot. With this, we expect to capture the sense that whenever other than a food type is a value and has a slot of *food_type* then it is punished. The second formula only relates the *value/2* and *slot/3* predicates together, this will punish the cases when a word is not a value and it was assigned a slot. The other two formulae are the case for the last element of a value of a slot.

8.3 Experiment and results

We perform three experiments: two of them with the model presented in the previous section, and a baseline:

1. We create a baseline system based on a word spotting system.

2. We test our model with gold transcriptions of the TownInfo corpus.

3. We test our model with automatic transcriptions of the TownInfo corpus.

The first experiment creates a baseline system with which to compare our system. We did not opt for Hidden Vector State [HVS, He and Young, 2005] as a baseline system because of semantic representation: in particular, the case of triplets which do not take as a value an element of the utterance would have required major changes in the HVS formulation. Instead the Keyword parser of the MATCH[1] system was used.

The following two experiments attempt to measure the performance of the ML model proposed in the previous chapter. In particular, we are interested into measuring the impact of switching from human transcriptions to automatic transcriptions. For these experiments we used 20 iterations over the development set in order to identify the best model. Following this, the identified model was applied to the testing corpus afterwards.

8.4 Results

Table 8.1 presents the results for our three set of experiments. The table reports the global and exact match metrics. We observe that the ML model performed better than the baseline. This was expected, since the baseline system uses a vocabulary which is based only on the training set, and the approach does not have the means to reach out for the context of the utterances. When analysing the results notice that for this corpus the exact match metric is higher that that of the global. This was not the case for the ATIS and Communicator corpus (see 5.4.3). This is because the TownInfo corpus has a low number of slots per utterance (see section A.3), with a median of 1. This is because utterances such as *yes* and *no* are common.

We notice that the performance of the model drops for the ASR transcriptions. This is in part because the ASR transcription has more cases of *triplet/3* predicates. This means there are more slots for which their value is not part of the utterance. There are 462 in the gold transcriptions, version and 934 in the ASR transcription version (see section A.3). This is because the ASR proposes words which are not part of the vocabulary of the system. The word error rate (WER%) for such ASR transcriptions is

[1] http://www.match-project.org.uk

Experiment	Global	Exact match
Baseline/KeyWord parser	67.06%	64.32%
Gold transcriptions/ML model	87.80%	90.49%
ASR transcriptions/ML model	62.19%	67.36%

Table 8.1: Results for the TownInfo corpus. Baseline system obtained with the KeyWord Parser. Transcriptions systems obtained with the ML model presented in section 8.2.

Experiment	Global	Exact match
Baseline/KeyWord parser	83.27%	90.27%
State of the art using gold transcriptions*	95.69%	N/A
State of the art using ASR transcriptions*	88.58%	N/A
ML with gold transcriptions	94.82%	95.35%
ML with ASR transcriptions	68.13%	72.77%

Table 8.2: Results for the TownInfo corpus. Baseline system obtained with the KeyWord Parser. State-of-the-art results from [Mairesse et al., To appear.] which were obtained with a different version of the corpus TownInfo. ML results obtained with the ML model presented in section 8.2.

41.96% which is high for a working system. Most of the errors (deleted *mass*) comes from one word utterances which are missrecognised and therefore such cases get 100% WER rate.

Table 8.2 shows the performance of the different systems when we only evaluate those utterances which have values that are part of the utterance (that is, they do not have a *triplet/3* predicate). This produces a performance that is approximately 5% higher for the global score and for exact match. This confirms our hypothesis that slots with values which are not part of the utterance are harder to deal with. In the table 8.2 we also include the state-of-the-art results of a new discriminative approach [Mairesse et al., To appear.]. The results are directly comparable since these were obtained with an alternative version of the TownInfo corpus, which only consists of slots whose values are present in the utterance. It is also not directly comparable with ours in that this approach uses a gazetteer, which could explain why their approach performs quite well with the ASR transcriptions. Additionally, the low performance of our model could be related to the performance to the ASR system.

8.5 Discussion

In this chapter we have presented a semantic processing module for the TownInfo system [Georgila et al., 2005b] developed in the Markov Logic framework [ML, Richardson and Domingos, 2006]. This module is a working module and was developed within a months work. Additionally, the Towninfo corpus used in this chapter was labelled with its current version of semantic representation in less than 20 hours. The development of the module presented the following challenges: it was a small corpus, there were no extra resources available and the semantic representation was tailored to the dialogue system. We designed an ML model to handle the semantic representation in a similar way to that in which we created a new model for the Semantic Role Labelling (SRL) task in the previous chapter. Additionally, we tested its performance with the small corpus and no extra resources.

We found that for this corpus, the performance can reach an acceptable result for the gold transcriptions. The main problem for the model is the triplets which have a value that is not part of the utterance. For instance, *i want a curry restaurant* and *i want an Indian restaurant* both have a slot *restaurant* with a value *indian*, however in the first utterance the value is not present in the utterance as it is in the second utterance. The first case is harder to handle for our model since it has to propose a whole *triplet/3* predicate for such cases. Table 8.3 presents the F1-scores for the ML predicates. The model is good in recognising when it has a slot value (*hasSlot/0* and *hasTriplet* scores are higher then 90%). Its performance on identifying speech acts is good as well: this is because there are only 4 types of speech acts. It is not particularly good at identifying the slots, but it still produces a reasonable performance. However, we can see that the performance is worst for the *triplet/3* predicate, and this was a consistent finding for the experiments performed with the ML model. The situation worsen when dealing with ASR transcriptions; the *triplet/3* predicate performance dropped to 34%. This is because the number of values which were not present in the utterance increased. In the original corpus we had 719 utterances, of which 217 had a value which was not part of the utterance. However, when working with the ASR transcription there were 390.

In summary, the semantic processing module performance is comparable with state-of-the-art approaches which use similar semantic representations and gold transcriptions. However, our model presents some difficulties with ASR transcriptions. We plan to do further research to investigate how to increase the performance. Our main directions of interest include using the *n*-best list of ASR transcriptions, incorporating

Predicate	F1-score
empty/0	97.9%
sa/2	94.5%
slot/3	86.9%
value/1	95.5%
hasTriplet/0	93.3%
triplet/3	72.00%

Table 8.3: F1-scores for the ML predicates of the TownInfo model.

extra information, and/or experimenting with factoring the ML model into one model for slots with values which are part of the utterance and a different model for the slots which are not part of it.

In the following chapter we will conclude the thesis. We present a summary of the three main topics which have been discussed in three previous chapters.

Chapter 9

Conclusion

In this chapter we discuss the main contributions of this work and the future directions of work.

9.1 Contributions

In this work we test the adequacy of Markov Logic [ML, Richardson and Domingos, 2006] for performing semantic processing, in particular for the Spoken Language Understanding (SLU) module of a dialogue system and the Semantic Role Labelling (SRL) task. In order to achieve this we focused on three aspects. Firstly we compared the ML with state-of-the-art approaches to SLU. In particular we focused on a setting that included minimal resources. Secondly, we tested the ML framework with a semantic representation that was more complex than the frame-based semantic representation typically used in dialogue systems. Thirdly, we performed a real world case scenario for the development of a SLU module for a dialogue system. In following subsections we present the contributions to,and insights from each of these aspects.

9.1.1 ML and state-of-the-art approaches

In order to compare the ML framework with state-of-the-art approaches, we created an equivalent ML model to the Hidden Vector State [HVS, He and Young, 2005]. In particular, we focused on the case when the system developer did not have access to extra resources such as gazetteers (see Chapter 5). In this setting we showed that a simple model could improve the results reported for the HVS model even though the reported results were obtained when using a gazetteer. We also showed that our model

performed better than our implementation of the HVS and MaxEnt local models when using minimal resources. The results of this model were published in Meza-Ruiz et al. [2008a].

Once we had established the performance of the ML model using a similar setting to the HVS model, we focused on three questions that were raised with the previous model (see Chapter 6). For the first question we revised the preprocessing step of lexical substitution. This step consisted of joining some of the words of the utterance, for instance, *new york* into *new_york*. We looked into avoiding such a step, so that the involvement of the dialogue system developer was minimal. To this end we used the BIO notation to label the words which were values of a slot. For the second question we investigated different architectures of our ML model. In particular, we created the *one layer* and *two layer* models. The *one layer* model treats slot and BIO notations as a single label, while the *two layer* model treats the slot and BIO as independent labels. For the third question we investigated the use of extra information. In particular we used off-the-shelf resources to extend the labelling and the gazetteer information. Previous results from the two layer model were published in Meza-Ruiz et al. [2008b].

We found that avoiding lexical substitution makes the task more difficult. The global score dropped significantly, and it was not possible to improve the score of the model without lexical substitution unless the gazetteer information was used. However, for the case of the exact match scores, the models were able to improve these scores by adding extra information such as syntactic chunks or by changing the model from *one layer* to *two layer*. This means that although the models were not able to recover as many slots as the original model, the ones which were recovered produced more adequate semantic frames. This signals the adequacy of the *two layer* model, which treats the slot and BIO label independently. However, we noted that this model increased the complexity of the ML model, and this had an impact on training time. We also found that using information provided by the gazetteer was the best way of improving the performance. This is consistent with current models for Spoken Language Understanding which rely on the use of gazetteers [Mairesse et al., To appear.]. However, in this thesis we explored the case where the system developer did not have access to such resources. We found that the syntactic chunks, in particular the head of chunks, were informative for the semantic processing task, and produced an significant improvement in performance.

9.1.2 Semantic Role Labelling

We tested the adequacy of the ML framework for handling semantic representations which are more complex than the frame-based representations. For this we created a ML model to carry out Semantic Role Labelling (SRL). We found that with the ML framework we could develop a state-of-the-art SRL system. This system was ranked second in the open track of the CoNLL-08 shared task. In this setting the system had to label verbal and nominal predicates. By systematically testing the structural constraints (these are the formulae which link the different hidden predicates) we found that the Bottom-up model performed the best. This model resembles a typical SRL architecture, where the output of the lower stages of the SRL are passed as input to the higher stages. The results of this system were published in Riedel and Meza-Ruiz [2008].

Additionally, we wanted to test the effect of the joint modelling of the ML framework on the SRL task. To this end, we further developed our SRL system for labelling verbal predicates. This resulted in an improved system despite the fact that the quality of the dependency parsing labels was worse than in our previous experiments. We confirmed our finding that the Bottom-up model performed better. We also found that the joint aspect of our system was helpful, since a directly comparable pipeline system performed worse, and required more time during training and labelling, besides the struggle to set it up properly. We also showed that our decision of identifying potential arguments independently of the predicate they belonged to offered a benefit in performance. The results of this system were published in Meza-Ruiz and Riedel [2009].

9.1.3 Real world case study

We developed a SLU module for a dialogue system using the ML framework. This was challenging since the development had a small corpus, there were no extra resources available, and the system had a tailored semantic representation. In particular, the experience acquired during the experimentation with minimal resources was invaluable to this development, since we knew what to expect in terms of performance. We also applied some of the knowledge acquired during the development of the SRL system. This was because the semantic representation required the use of different hidden predicates, and we needed them to be joined together. To tackle this we applied similar structural constraints than to those used in the SRL system.

Our resulting system reached a good performance. However, we identified that the ML model found slots which were not related to a value in the utterance to be difficult to handle by the ML model. This situation worsened for automatic transcriptions. With respect to this, in the following section we present some directions for further research with this respect.

9.2 Future work

There are three issues around which we plan further research on the ML framework:

- Our current results for the SLU task and the SRL task although reasonable when compared with state-of-the-art systems (e.g., He and Young [2005]; Mairesse et al. [To appear.]), are still lower than the best systems on the field. For instance, Zettlemoyer and Collins [2007] propose an Online CCG parser. Their system obtained a 95% of the global score, while our best model reaches 92%. Although our approach is not not directly comparable, since their approach uses a different semantic representation, we have not identified the main reason of such difference. Other approaches that are closer to ours reach a 93% global score (e.g., [Mairesse et al., To appear.; Ye and Young, 2006]). Another example is the SRL system proposed by Johansson and Nugues [2008], which gives a higher score for the SRL system with a semantic dependency score of 86% for the PropBank while our system currently reaches 82%. We understand some of the causes for the gaps in performance. In SLU approaches, the lexical substitution is integral to the approaches, but in our models we detached this from the pipeline and let the ML framework model it. In SRL the quality of the dependency parses has an impact on the SRL labelling. However, this difference may not be the only cause of the performance gap: it could also be the training regime used by the ML framework. New settings and techniques for training have to be tested in the future to push the performance of the current ML. For instance, there is available to the implementation of the ML system we use a new set of loss functions which we are testing: these have shown an improvement in the performance.

- Our work with SRL aimed to show that the ML framework was adequate for the task. Our goal was to test the ML for semantic representations which are more complex than the frame-based representations traditionally used, but they are

shallow enough that they are attractive to the dialogue system field. Our results point to the fact that ML is able to deal with these semantic representations. However, now further research has to be done into transferring those models to the dialogue system fields. In particular, we must test those models in similar circumstances to those presented here for the SLU task: using a small corpus, with limited resources or generated from off-the-shelf resources.

- While working with automatic transcriptions in our experiments, we noted that the performance drastically dropped. This could be attributed to the quality of the ASR module, however, further research has to be done to incorporate into our methods the traditional techniques for dealing with this issue, such as: *n*-best lists and confidence scores. So far, it is not clear how to incorporate this information into the ML framework so that the Markov Networks take into account the differences among the transcription hypotheses.

In general, further research has to be done on reducing the workload of the dialogue system developer for the development of SLU modules. For instance, we need to explore semi-supervised or unsupervised techniques of natural language processing, and consider using them in SLU development (e.g., Ye and Young [2006]; Zhou and He [2008]). Or the techniques which allow transfer of knowledge so that a model currently used in a system can be transferred into a newer system: in this sense the adoption of semantic roles as semantic representations can play an important role in the dialogue system task (e.g., He and Young [2006]). In summary, this thesis contributes new methods and results for robust semantic processing and quantifies their usefulness in developing systems using minimal resources.

Appendix A

Corpora information

In this appendix we present the elements which compose the frame-based semantic representations of the corpora Air Traffic Information System [ATIS, Dahl and et al., 1994], Communicator [Bennett and Rudnicky, 2002] and semantic triplets of the TownInfo [Lemon et al., 2006a].

A.1 ATIS corpus

The ATIS corpus version developed by He and Young [2005] labels the goals and slot-values in the frame-based representation. The following figures and tables presents some aspects of the corpora:

- Table A.1 presents some basic statistics of the corpus.
- Table A.2 enumerates the goals and slots of the corpus.
- Figure A.1 presents the histogram of the lengths of the utterances.
- Figure A.2 presents the histogram of the number of slots per utterance.
- Figure A.3 presents the histogram of number of types of goals in the corpus.

For an introduction to the corpus see section 3.1.

A.2 Communicator corpus

The Communicator corpus version developed by He and Young [2005] labels the goals and slot-values in the frame-based representation. The following figures and tables present some aspects of the corpora:

Average length	11.21
Median length	12
Average number of slots	3.38
Median number of slots	4
Number of token words	50,250
Number of type words	897
Number of token goals	4,481
Number of type goals	21
Number of token slots	15,171
Number of type slots	95
Number of token slots of length 1	3,155
Number of types slots of length 1	50
Number of token slots of length 2	12,016
Number of types slots of length 2	45

Table A.1: Counts of the elements which compose the training part of the ATIS corpus. The main elements are: words, goals and slots.

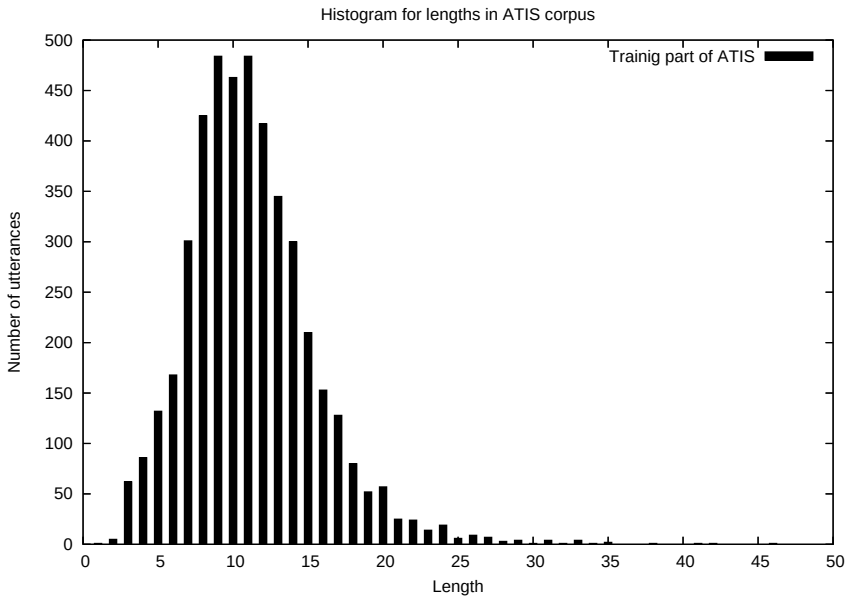


Figure A.1: Histogram for the lengths of utterances of the training part of the ATIS corpus. Most lengths are around 11 words per utterance.

Goals:

FLIGHT [3873]
GROUND_SERVICE [213]
AIRLINE [99]
AIRFARE [88]
ABBREVIATION [86]
AIRCRAFT [47]
FLIGHT_AIRFARE [14]
AIRPORT_DISTANCE [10]
FLIGHT_NO [9]
FLIGHT_TIME [8]
CAPACITY [7]
AIRPORT [7]
GROUND_FARE [5]
RESTRICTION [3]
FLIGHT_AIRCRAFT_FLIGHT_NO [3]
CITY [3]
AIRLINE_FLIGHT_NO [2]
MINIMUM_CONNECT_TIME [1]
LOCATION [1]
FLIGHT_AIRLINE [1]
DISTANCE [1]

Slots:

FROMLOC.CITY_NAME [3867]
TOLOC.CITY_NAME [3855]
DEPART_DATE.DAY_NAME [828]
AIRLINE_NAME [623]
DEPART_TIME.PERIOD_OF_DAY [607]
DEPART_DATE.DAY_NUMBER [374]
DEPART_DATE.MONTH_NAME [359]
COST_RELATIVE [315]
ROUND_TRIP [309]
DEPART_TIME.TIME [299]
FLIGHT_MOD [296]
DEPART_TIME.TIME_RELATIVE [296]
CITY_NAME [288]
CLASS_TYPE [227]
STOPLOC.CITY_NAME [225]
ARRIVE_TIME.TIME_RELATIVE [174]
FLIGHT_STOP [148]
ARRIVE_TIME.TIME [143]
AIRLINE_CODE [118]
ARRIVE_TIME.PERIOD_OF_DAY [99]
DEPART_DATE.DATE_RELATIVE [78]
TOLOC.STATE_NAME [76]
DEPART_DATE.TODAY_RELATIVE [76]
TOLOC.STATE_CODE [75]
DEPART_TIME.PERIOD_MOD [75]
FLIGHT_NUMBER [69]
FARE_BASIS_CODE [68]
FLIGHT_TIME [59]
OR [57]
ARRIVE_DATE.DAY_NAME [57]
MEAL_DESCRIPTION [53]
DOWNTOWN [53]
FARE_AMOUNT [50]
MEANING [45]
FROMLOC.STATE_CODE [43]
DEPART_TIME.TIME_RANGE [40]
TRANSPORT_TYPE [39]
MEAL [37]
FLIGHT_DAYS [37]
CONNECT [37]
FROMLOC.STATE_NAME [35]
ARRIVE_DATE.MONTH_NAME [35]
ARRIVE_DATE.DAY_NUMBER [35]
AIRPORT_NAME [35]
FROMLOC.AIRPORT_NAME [33]
TOLOC.AIRPORT_NAME [31]
ARRIVE_TIME.TIME_RANGE [28]
MOD [27]
AIRPORT_CODE [25]
AIRCRAFT_CODE [25]
RETURN [22]
RESTRICTION_CODE [21]
DEPART_TIME.END_TIME [21]
DEPART_DATE.YeAR [21]
ARRIVE_TIME.START_TIME [20]
DEPART_TIME.START_TIME [19]
ARRIVE_TIME.END_TIME [19]
TOLOC.AIRPORT_CODE [18]
FROMLOC.AIRPORT_CODE [12]
ARRIVE_DATE.DATE_RELATIVE [11]
LOCATION [8]
CODE [8]
STATE_CODE [6]
MEAL_CODE [6]
DEPART_DATE.TIME [6]
STOPLOC.STATE_CODE [5]
AND [5]
ABBREVIATION [5]
MANUFACTURER [4]
ARRIVE_TIME.PERIOD_MOD [4]
TOLOC.COUNTRY_NAME [3]
THEN [3]
STATE_NAME [3]
RETURN_DATE.DATE_RELATIVE [3]
PROPULSION [3]
NAME [3]
BOOKING_CLASS [3]
ARRIVE_TIME.TIME_MOD [3]
ARRIVE.CITY_NAME [3]
NO [2]
DAYS_CODE [2]
ARRIVE_DATE.TODAY_RELATIVE [2]
AIRLINE [2]
TIME_ZONE [1]
TIME [1]
MINIMUM_CONNECT_TIME [1]
FLIGHT_STOP.MOD [1]
ECONOMY [1]
DUMMYE [1]
DISTANCE [1]
DISCOUNT [1]
DEPART_TIME.TIME_MOD [1]
DEPART_DAY.DAY_NAME [1]
AIRFARE_CODE [1] AIRCRAFT_MOD [1]

Table A.2: List of goals and slots in ATIS corpus. The number between square brackets represents the frequency of the slot in the training part of the ATIS corpus.

aircraft_code, airline_code, airline_name, airport_code, airport_name, city_code, state_name, city_name, class_type, country_name, day_name, days_code, fare_basis_code, manufacturer, meal_code, meal_description, month_name, restriction_code, state_code, time, transport_type, year, cost_relative, day_number, flight_mod, flight_stop, flight_time, period_of_day, round_trip, today_relative, and downtown

Table A.3: Lexical classes in the gazetteer of the ATIS corpus

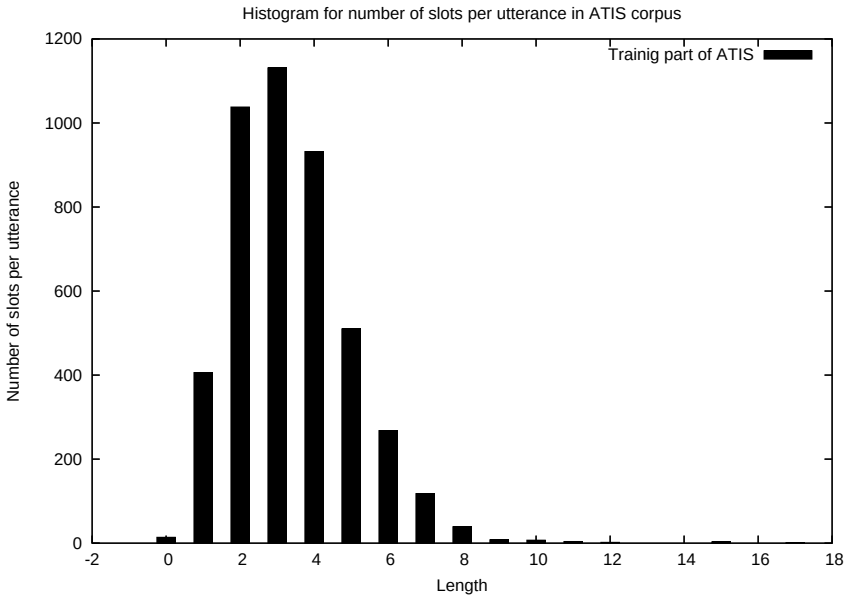


Figure A.2: Histogram of number of slots per utterance of the training part of the ATIS corpus. Most utterances have 2, 3 or 4 slots.

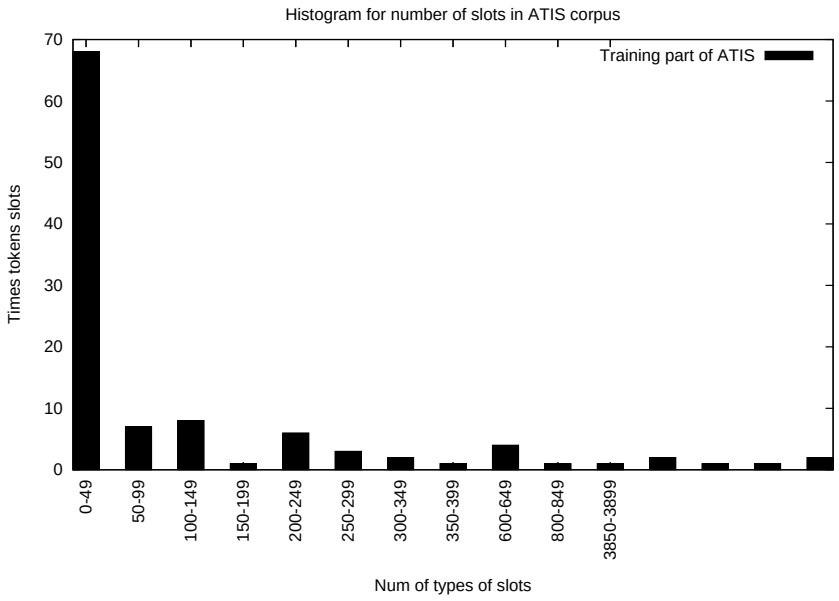


Figure A.3: Histogram of number of types of slots of the training part of the ATIS corpus. Most slot types occur between 0 to 49 times. Notice that two slots occur between $3,850$ and $3,899$ times, these are `FROMLOC.CITY_NAME` and `TOLOC.CITY_NAME`.

- Table A.4 presents some basic statistics of the corpus.
- Table A.5 enumerates the goals and slots of the corpus.
- Figure A.5 presents the histogram of the lengths of the utterances.
- Figure A.6 presents the histogram of the number of slots per utterance.
- Figure A.7 presents the histogram of number of types of slots in.

For an introduction to the corpus see section 3.2.

Average length	4.417
Median length	7
Average number of slots	3.056
Median number of slots	2
Number of token words	56,095
Number of type words	1,027
Number of token goals	12,702
Number of type goals	20
Number of token slots	26,117
Number of type slots	108
Number of token slots of length 1	18,651
Number of types slots of length 1	54
Number of token slots of length 2	7,466
Number of types slots of length 2	54

Table A.4: Counts of the elements which compose the training part of the Communicator corpus. The main elements are: words, goals and slots.

A.3 TownInfo corpus

The TownInfo corpus uses triplets of speech act, slot and values as a semantic representation [Georgila et al., 2005a]. There are two different versions of the corpus. One contains gold standard transcriptions and the other ASR transcriptions. The following figures and tables presents some aspects of the corpus:

- Table A.6 presents some basic statistics of both version of the corpus.

Goals:

DATE_TIME [5577]
AIR [2651]
CITY [1721]
AIRLINE [743]
CAR [456]
HOTEL [403]
DESTINATION [390]
RETURN_TIME [243]
DEPART_TIME [217]
ORIGIN [206]
RETURN [48]
FARE [28]
HOTEL/CAR [5]
ARRIVE_TIME [4]
FLIGHT_TIME [3]
STOP [2]
DEPART [2]
TRAIN [1]
QUERY [1]
AIRPORT [1]

Slots:

DAY_NUMBER[3378]
MONTH_NAME[3371]
PERIOD_OF_DAY[3023]
TOLOC.CITY_NAME[2306]
TIME[2076]
CITY_NAME[2069]
FROMLOC.CITY_NAME[1696]
AIRLINE_NAME[875]
RETURN.DAY_NUMBER[445]
STATE_NAME[443]
RETURN.MONTH_NAME[438]
DAY_NAME[434]
TIME_RELATIVE[378]
TOLOC.STATE_NAME[344]
RENTAL_COMPANY[322]
DEPART.PERIOD_OF_DAY[293]
HOTEL_NAME[274]
RETURN.PERIOD_OF_DAY [266]
FLIGHT[239]
NEGATIVE[235]
DEPART.MONTH_NAME[223]
DEPART.DAY_NUMBER[223]
DATE_RELATIVE[211]
DEPART.CITY_NAME[209]
TODAY_RELATIVE[183]
FROMLOC.STATE_NAME[146]
RETURN.CITY_NAME[134]
CAR[125]
HOTEL[123]
FLIGHT_MOD[119]
YeAR[117]
RENTAL_TYPE[105]
TOLOC.COUNTRY_NAME[101]
DEPART.TIME[89]
RETURN[83]
ROUND_TRIP[76]
AIRPORT_NAME[75]
HOTEL_LOC[53]
FROMLOC.AIRPORT_NAME[52]
DEPART.DAY_NAME[50]
PERIOD_MOD[49]
RETURN.DATE_RELATIVE[47]
COST_RELATIVE[43]
RETURN.DAY_NAME[42]
COUNTRY_NAME [42]
TOLOC.AIRPORT_NAME [41]
ARRIVE.CITY_NAME [37]
DEPART.TIME_RELATIVE [30]
ARRIVE.TIME [24]
LEG_NUM [23]
ARRIVE.PERIOD_OF_DAY [22]
DEPART.DATE_RELATIVE [18]
RETURN.YeAR [17]
DEPART.YeAR [17]
ARRIVE.TIME_RELATIVE [16]
TIME_END [15]
RETURN.PERIOD_MOD [14]
DEPART.STATE_NAME [14]
ARRIVE.DAY_NUMBER [14]
FLIGHT_STOP [13]
ARRIVE.MONTH_NAME [13]
TIME_START [12]
DEPART.PERIOD_MOD [12]
AIRLINE [10]
FARE [9]
FROMLOC.AIRPORT_CODE [8]
DEPART.TODAY_RELATIVE [8]
STOPLOC.CITY_NAME [7]
RETURN.TIME_RELATIVE [7]
RETURN.STATE_NAME [7]
DEPART.AIRPORT_NAME [7]
HOLIDAY [6]
FARE_AMOUNT [5]
RETURN.TIME [4]
DEPART_TIME [4]
DEPART [4]
CLASS_TYPE [4]
ARRIVE.STATE_NAME [4]
FLIGHT_TIME [3]
DEPART.TIME_START [3]
DEPART.TIME_END [3]
DATE [3]
ARRIVE.DAY_NAME [3]
TOLOC.AIRPORT_CODE [2]
NEGAIVE [2]
HOTEL_DURATION [2]
FLIGHT_OPTION [2]
DESTINATION [2]
DATE_ORDINAL [2]
TOLOC.CONTINENTAL_NAME [1]
RETURN.AIRLINE_NAME [1]
OR [1]
NEGATVE [1]
MEAL [1]
FROMLOC.COUNTRY_NAME [1]
FLIGHT_NUMBER [1]
DEPART.HOLIDAY [1]
DEPART.COUNTRY_NAME [1]
CONTINENTAL_NAME [1]
ARRIVE.YeAR [1]
ARRIVE.TODAY_RELATIVE [1]
ARRIVE.DATE_RELATIVE [1]
ARRIVE.COUNTRY_NAME [1]
ARRIVE.AIRPORT_NAME [1]
ARRIVE [1]
AIRPORT [1]
AIRLINE_CODE [1]

Table A.5: List of goals and slots in the Communicator corpus. The number between square brackets represents the frequency that the slot happened in the training part of the Communicator corpus.

	Gold transcriptions	ASR transcription
Average length	3.60	3.80
Median length	6	6
Average number of triplets	1.13	1.14
Median number of triplets	1	1
Number of token words	6,160	6,436
Number of type words	343	389
Number of token speech acts	1,933	1,933
Number of type speech acts	4	4
Number of token slots	1,933	1,933
Number of type slots	17	17
Number of token slots with value in the utterance	1,471	999
Number of types slots with value in the utterance	16	14
Number of token slots with value not in the utterance	462	934
Number of types slots with value not in the utterance	43	13

Table A.6: Statistics for the training part of the TownInfo corpus

- Table A.7 enumerates the goals and slots of the corpus.
- Figure A.9 presents the histogram of the lengths of the utterances.
- Figure A.10 presents the histogram of the number of triplets per utterance.
- Figure A.11 presents the histogram of number of types of slots in.

For the first table, we differentiate among both corpus. However for the rest of the corpora the figures are identical, since this information is based on the semantic representation and not the transcription. For an introduction to the corpus see section 3.3.

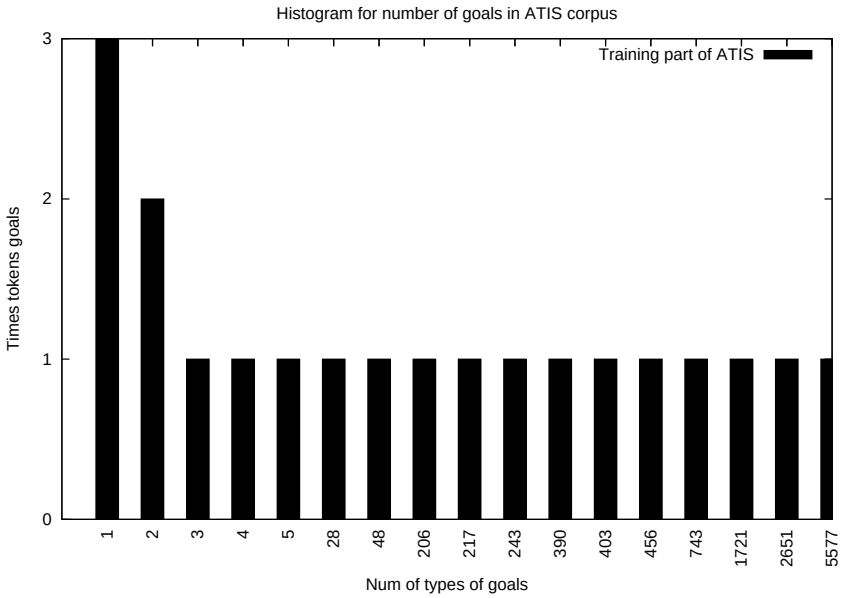


Figure A.4: Histogram of number of types of goals of the training part of the ATIS corpus.

Speech acts:

prov_info [1863]
wh_question [58]
yn_question [10]
prov_neg [2]

Slots:

answer [574]
task [322]
region [275]
price [190]
location [184]
stars [102]
bar_type [77]
request_quit [71]
food_type [69]
address [23]
phone_number [18]
result_choice_number [8]
music_type [8]
start_over [7]
near [2]
music [2]
request [1]

Table A.7: List of goals and slots in TownInfo corpus.

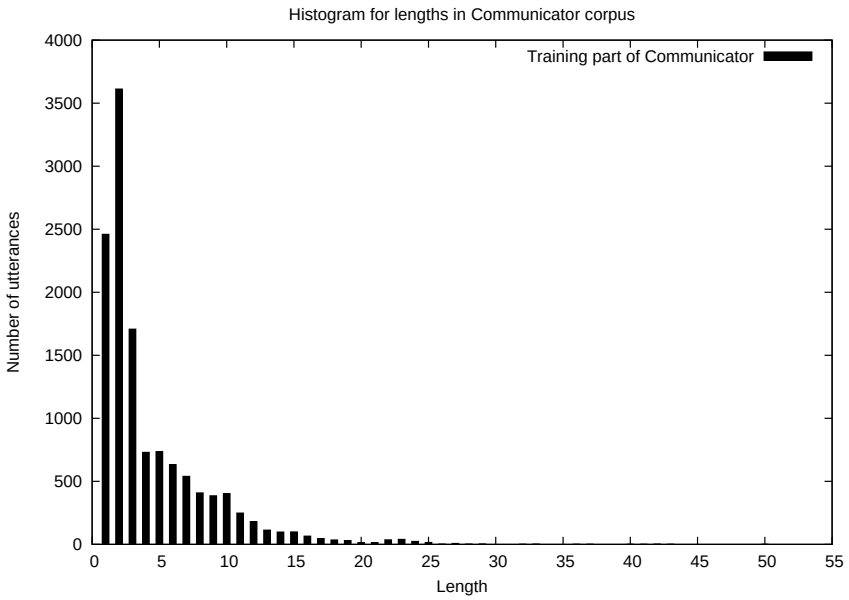


Figure A.5: Histogram for the lengths of utterances of the training part of the Communicator corpus. Most utterances are of length 2.

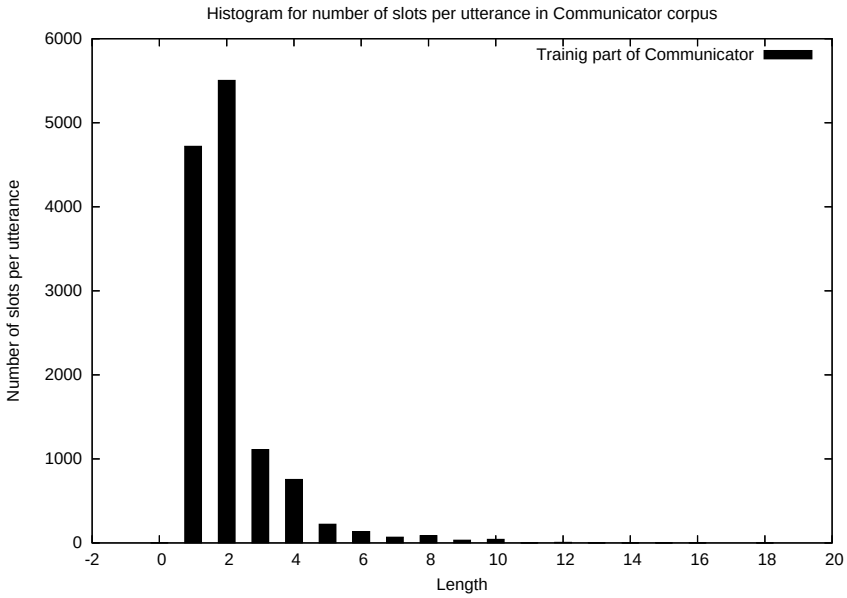


Figure A.6: Histogram of number of slots per utterance of the training part of the Communicator corpus. Most utterances have 1 or 2 slots.

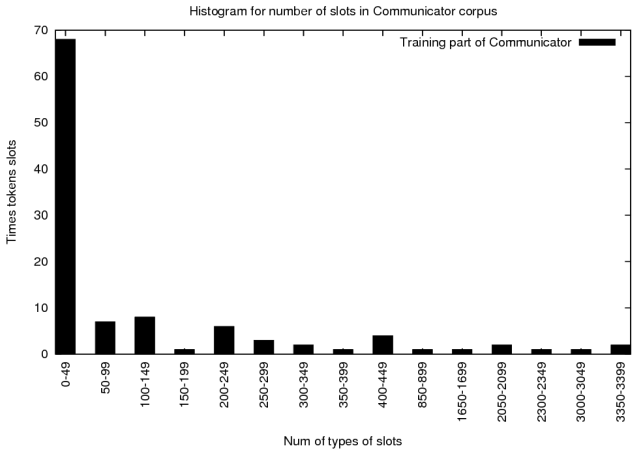


Figure A.7: Histogram of number of types of slots of the training part of the Communicator corpus. Most slot types occur between 0 to 49 times. Notice that 7 slots occur more than $1,000$ times.

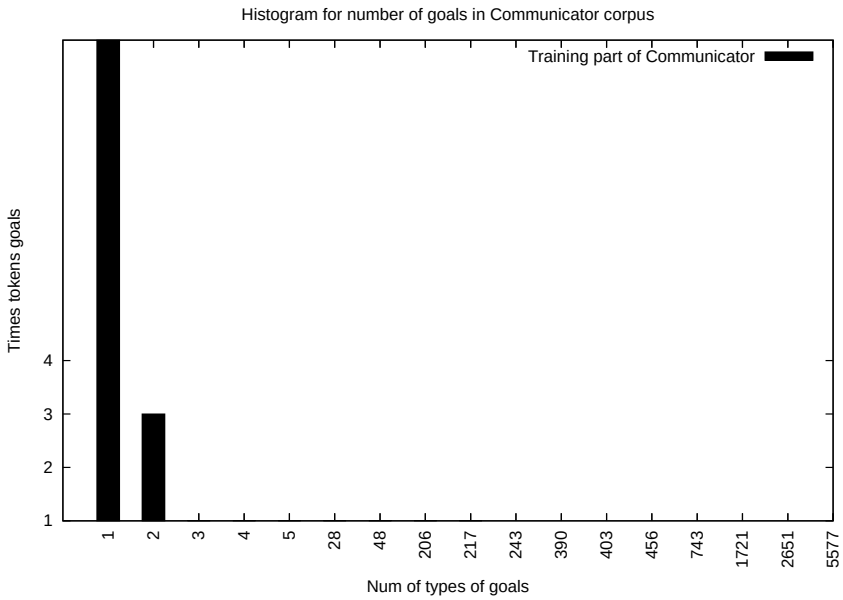


Figure A.8: Histogram of number of types of goals of the training part of the Communicator corpus

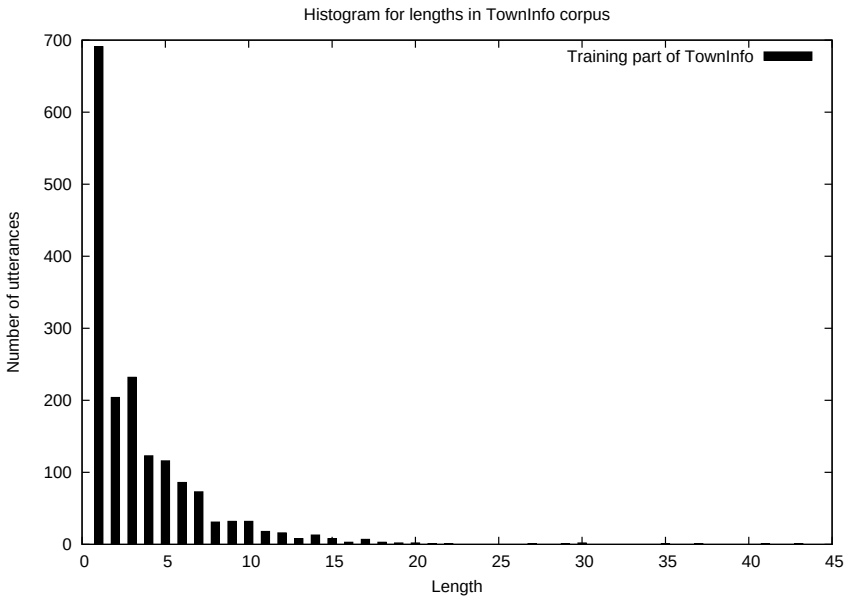


Figure A.9: Histogram for the lengths of utterances of the training part of the TownInfo corpus. Most utterances are of length 1, since there are many *yes* and *no* utterances.

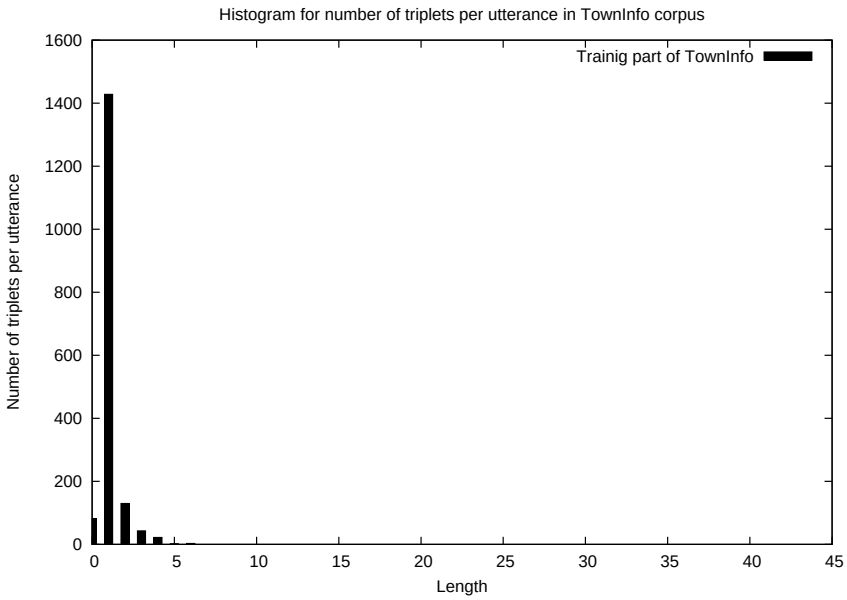


Figure A.10: Histogram of number of triplet per utterance of the training part of the Communicator corpus. Most utterances have 1 triplet.

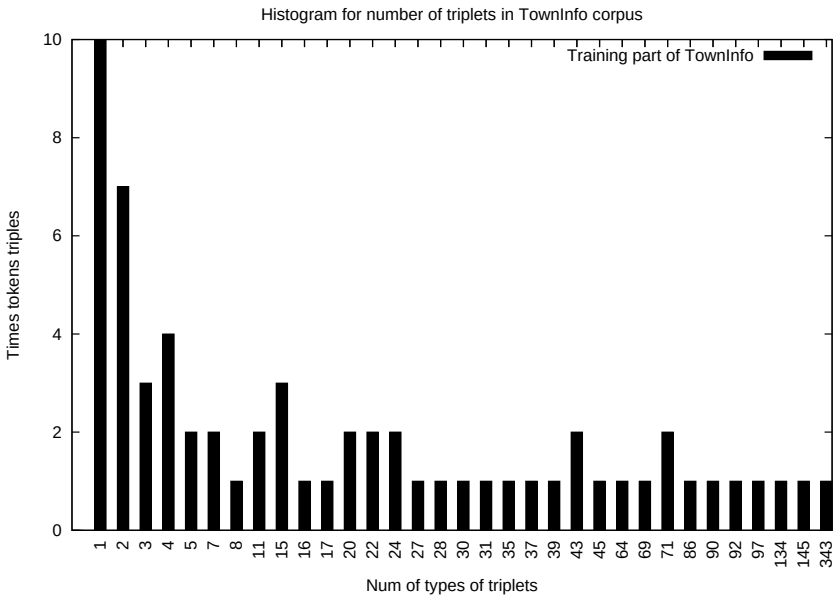


Figure A.11: Histogram of number of types of triplets of the training part of the TownInfo corpus. Most triplets types occur only 1 time. In particular the triplet for *yes* which occurs 343 times.

Appendix B

Replication experiments

In this appendix we present a set of experiments which aim to replicate the results reported in the literature for the Hidden Vector State [HVS, He and Young, 2005], in particular the ones presented in Table 5.3 from section 5.4. With these experiment we aim to show the adequacy of our implementation of the HVS model to be used as a baseline. We call these experiments the *replication* experiments. A comparison of the result of the replication experiments with the rest of experiment involving HVS and ML with minimal resources is presented in section 5.5.

B.1 The implementation

In subsection 5.1.3 we explained the main difference in the preprocessing steps between our implementation and the original implementation by He and Young [2005]. As explained, the main difference is that our implementation automatically infers the constraints for the EM algorithm, while the original implementation relies in the abstract semantic representation.

Besides this difference our implementation in two aspects, as well: we perform floor-value smoothing for the probabilities and we include an initial probability for the vector states. The smoothing technique was chosen for its convenience during implementation. The inclusion of an initial probability was made to handle cases where a slot-value appears in the first word of the utterance. In these cases, the HVS model requires some element already present in the stack, otherwise it is unable to recognize this word as a value of a slot.

Finally, there is another point of difference between our implementation and that of He and Young [2005] which is more of a post-processing step. Their implementation

needs some "heuristics" to recover the slot-values from the labelling. They have to analyse each vector state and extract the concepts which are relevant depending on a set of rules conditioned on the presence of concepts. For our implementation, the heuristics consist of taking from the identified stack the tail concepts which are a valid slot. For instance, in Figure 4.3 in section 5.1 the stack for the word *chicago* is:

$$[FLIGHT, ARRIVE\_TIME, TOLOC, city\_name]$$

To recover the valid slot, in the case of our implementation we only need to recover the last two elements of the stack which forms the slot:

$$TOLOC.CITY\_NAME = chicago$$

In summary, our implementation automatises the whole training and labelling process for the dialogue system developer. In theory, the user only has to provide the corpus of pair of utterance with their frame-based semantic representations. The elements of the HVS model are defined by this corpus, the training and labelling are done without any extra labellings.

B.2 Replication experiments

We performed the following experiments to verify that our implementation could replicate the previous state of the art performance. For this reason, we tested our implementation under a similar setup to that described in He and Young [2006]. We modelled the slots that are not a `goal` slot with the HVS model, and we used a different procedure for identifying the `goal` concept. In particular, for the results we will show in this chapter the `goal` slot was identified using a Maximum Entropy classifier with the following features:

- Number of times a word occurs in an utterance.
- Number of times a bi-gram occurs in an utterance.

Table B.1 shows our results. Besides the global score, we report the average and the exact match scores which have not been measured before for the HVS approach. We tested our implementation using both the ATIS corpus and the Communicator corpus. As you can appreciate from the results, we were unable to precisely replicate the results of He and Young [2006] for the ATIS corpora. However, for the Communicator corpus

	Precision	Recall	F-score
ATIS $NOV93$			
Global	85.35%	78.72%	**81.90**%
Average	78.43%	74.12%	75.91%
Exact match	47.37%	44.20%	45.73%
Goal Acc	77.23%		
ATIS $DEC94$			
Global	86.55%	80.75%	**83.55**%
Average	83.90%	79.20%	81.17%
Exact match	45.12%	44.72%	45.12%
Goal Acc	77.53%		
Communicator			
Global	90.57%	88.02%	**89.28**%
Average	91.40%	89.37%	89.95%
Exact match	78.76%	76.89%	77.82%
Goal Acc	95.41%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table B.1: Results for replication experiments with the HVS model. Our results are not as good for the ATIS testing corpora. But, they are comparable with the communicator.

this was possible. This is a mixed result which unfortunately does not fully validate our implementation of the HVS model.

It is hard to identify the reason why our implementation falls short in performance compared with He and Young [2006] for the ATIS corpus. Our main hypothesis is that the abstract representations used in their experiments is better suited for capturing long distance dependencies. These representations were inferred from the original set of `SQL` queries of the ATIS corpora queries rather than from the slot-value corpus. This situation was not possible for the Communicator system, since there is no corpus labelled with `SQL` queries.

Analysing the set of *abstract semantic representations*, we found the following differences between these and the slot-values we used in our implementation:

- Some concepts are substituted for general ones. For instance `DEPART_TIME` and `DEPART_DATE` are replaced by `DEPART`.
- Some terminal slots are simplified. For instance `FROM_LOC.CITY_NAME=CITY_NAME` is replaced by `FROM_LOC=CITY_NAME`.
- The concept representing the `goal` was not always the root, but a node in the tree.

In order to check whether these differences could be translated into any improvement we eliminated the first one from our implementation. We replaced any semantic concept with a prefix `DEPART` or `ARRIVE` (e.g. `DEPART_TIME` is replace by `DEPART`) by exactly those prefixes. We chose to implement this because in our development set the slots which contained these semantic concepts with these prefixes were commonly ignored, generating a pattern of errors.

Table B.2 shows the results for this experiment. As you can see, for both test sets there is an improvement which is statistically significant when compared with the output of our previous result. It is important to notice that the exact match score improved by more than 5%. These results reinforce our belief that the inclusion of the *abstract semantic representation* is related to the good performance in the previous HVS results, and that these were more suitable for the long distance dependencies.

From the results with the development set we notice that slots corrected were the ones to which this modification was addressed. The recall of slots containing the concept prefix `DEPART` was improved. In particular, in our original implementation there were 37 errors involving the slot-values:

	Precision	Recall	F-score
ATIS $NOV93$ $\rho < 0.05$			
Global	84.77%	79.77%	**82.27**%
Average	77.82%	75.07%	76.20%
Exact match	53.35%	49.78%	51.50%
Goal Acc	77.23%		
ATIS $DEC94$ $\rho < 0.05$			
Global	86.36%	83.00%	**84.65**%
Average	83.48%	80.85%	81.89%
Exact match	54.23%	53.26%	53.74%
Goal Acc	77.53%		
State of the art results (HVS) Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%

Table B.2: Result of replication experiments with modification to the slots. In this results, we modified some of the slots in the frame-based semantic representation to match the abstract semantic representation of the original implementation of the HVS model.

- $DEPART\_TIME.PERIOD\_OF\_DAY = afternoon$
- $DEPART\_TIME.PERIOD\_OF\_DAY = morning$

Utterances containing a sequence of day name plus *afternoon* or *morning* were a problem. This was because in order to form these slots HVS has to pop 2 concepts from the previous stack (e.g., $DEPART\_DAY$ and DAY_NAME) and push 2 concepts (e.g., $DEPART\_TIME$, $PERIOD\_OF\_DAY$). However, this is not possible in the HVS framework, since only one element can be pushed. This was solved by the modification introduced above: both `DEPART_TIME` and `DEPART_DAY` become `DEPART`, therefore this sequence of day name and period of day only required to pop one concept, and to push one, a situation which is correct for the HVS framework.

In summary, the results presented so far can be interpreted as mixed since we replicated the state of the art results for the Communicator corpus but we could not reach the state of the art performance for the ATIS corpus. However, we show this difference could be explained by the addition of the abstract semantic representation in previous results with the ATIS corpus. We decided not to focus on creating an *abstract semantic representation* since this addition represents an extra labelling for the dialogue system developer, and we are interested in the case were the system developer has access to limited resources.

B.3 Discusion

In this appendix we have presented our efforts to replicate original performance of the Hidden Vector State [HVS, He and Young, 2005]. During this process we faced the problem that our implementation of the HVS model was unable to replicate all the previously reported results (see Table B.1). In particular, we could not replicate the ATIS corpora reported in He and Young [2006]. We investigated what could be the cause and we found that previous experiments relied on an extra labelling called *abstract semantic representation*. We modified our implementation with some of the considerations of this labelling and found an statistically significant improvement (see Table B.2).

On the other hand, our implementation was able to best replicate the results for the Communicator which was reported in the same paper by He and Young [2006]. For this reason, and considering that the use of abstract semantic representations consists in extra resources to which our system developer could not have access, we conclude

	Precision	Recall	F-score
ATIS $NOV93$			
Global	88.73%	87.44%	**88.08**%
Average	85.78%	84.75%	85.02%
Exact match	71.33%	69.42%	70.36%
Goal Acc	83.48%		
ATIS $DEC94$			
Global	86.18%	87.27%	**86.72**%
Average	85.53%	87.19%	86.01%
Exact match	69.48%	68.54%	69.00%
Goal Acc	84.04%		
Communicator			
Global	90.24%	92.21%	**91.22**%
Average	92.60%	93.73%	92.76%
Exact match	85.64%	83.64%	85.75%
Goal Acc	95.75%		
State of the art results Global score			
ATIS3 $NOV-93$	N/A	N/A	**90.30**%
ATIS3 $DEC-94$	N/A	N/A	**91.90**%
Communicator	87.20%	91.90%	**89.50**%

Table B.3: Replication experiments for discrete local model.

that our baseline results are reasonable to compare with. However we will consider them a weak baseline, and as we show in Chapter 5 we create a stronger baseline using a discriminative local model. Table B.3 presents the results of this local model when used in a similar setup to the replication experiments.

Appendix C

SRL local formulae

In this appendix we list the First Order Logic (FOL) local formulae for the Markov Logic [ML, Richardson and Domingos, 2006] models to perform Semantic Role Labelling [SRL, Márquez et al., 2008]. There is section for each hidden predicate of the model. We split the formulae for each hidden predicate into groups which are briefly explained at the start of each group. There are two models, the original which was used to obtain the results presented in section 7.4 and an extension of this model which was used to obtain the results in sections 7.5. The differences between the two models are explained at the start of each section and group of rules.

In order to simplify the number of formulae for some of the cases we put between squares brackets all versions of such rule, for instance:

$$lemma(p,+lp) \wedge role(p,a,+r) \wedge |p-a| = [0,1,2,]$$

should be interpreted as three rules:

$$lemma(p,+lp) \wedge role(p,a,+r) \wedge |p-a| = 0$$
$$lemma(p,+lp) \wedge role(p,a,+r) \wedge |p-a| = 1$$
$$lemma(p,+lp) \wedge role(p,a,+r) \wedge |p-a| = 2$$

The complete code for the formulae can be consulted in:

- http://code.google.com/p/thebeast/source/browse/#svn/mlns/conll08
- http://code.google.com/p/thebeast/source/browse/#svn/mlns/naaclhlt09

C.1 *isPredicate/1*

In the original formulae, the formulae does not contain the predicate *possiblePredicate/1*.

Current token: Captures the relation of the token with the context.

- $possiblePredicate(p) \wedge word(p, +wp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(p, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(p-1, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(p+1, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(p-2, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(p+2, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(p, +pp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(p-1, +pp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(p+1, +pp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(p-2, +pp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(p+2, +pp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge lemma(p, +lp) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge sform(p, +lf) \wedge isPredicate(p)$

Coarse POS tag: Relates the information of the Coarse POS tag around the predicate and argument tokens.

- $possiblePredicate(p) \wedge cpos(p+1, +cp1) \wedge cpos(p-1, +cp2) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge cpos(p+1, +cp1) \wedge cpos(p-1, +cp2) \wedge cpos(p+2, +cp3) \wedge cpos(p-2, +cp4) \wedge isPredicate(p)$

Dependency features: Relates the dependency structure of a predicate.

- $possiblePredicate(p) \wedge dep(p, \_, +d) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge dep(\_, p, +d) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(j, +pj) \wedge dep(p, j, +d) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(j, +pj) \wedge ppos(i, +pa) \wedge dep(p, j, +d) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge ppos(i, +pj) \wedge ppos(i, +pa) \wedge dep(i, j, \_) \wedge dep(j, p, +d) \wedge isPredicate(p)$
- $possiblePredicate(p) \wedge slemma(j, +pj) \wedge dep(j, j, +d) \wedge isPredicate(p)$

Frame features: Relates the subcategorisation frame.

- $possiblePredicate(p) \wedge frame(p, +f) \wedge isPredicate(p)$

C.2 *isArgument/1*

In the original formulae, the formulae does not contain the predicate *possibleArgument/1*.

Current token: Captures the relation of the token with the context.

- $possibleArgument(a) \wedge word(a,+wp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(a,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(a-1,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(a+1,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(a-2,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(a+2,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(a,+pp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(a-1,+pp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(a+1,+pp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(a-2,+pp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(a+2,+pp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge lemma(a,+lp) \wedge isArgument(a)$
- $possibleArgument(a) \wedge sform(a,+lf) \wedge isArgument(a)$

Coarse POS tag: Relates the information of the Coarse POS tag around the predicate and argument tokens.

- $possibleArgument(a) \wedge cpos(p+1,+cp1) \wedge cpos(p-1,+cp2) \wedge isArgument(a)$
- $possibleArgument(a) \wedge cpos(p+1,+cp1) \wedge cpos(p-1,+cp2) \wedge cpos(p+2,+cp3) \wedge cpos(p-2,+cp4) \wedge isArgument(a)$

Dependency features: Relates the dependency structure of an argument.

- $possibleArgument(a) \wedge dep(a,\_,+d) \wedge isArgument(a)$
- $possibleArgument(a) \wedge dep(\_,a,+d) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(j,+pj) \wedge dep(a,j,+d) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(j,+pj) \wedge ppos(i,+pa) \wedge dep(a,j,+d) \wedge isArgument(a)$
- $possibleArgument(a) \wedge ppos(i,+pj) \wedge ppos(i,+pa) \wedge dep(i,j,\_) \wedge dep(j,a,+d) \wedge isArgument(a)$
- $possibleArgument(a) \wedge slemma(j,+pj) \wedge dep(j,j,+d) \wedge isArgument(a)$

Frame features: Relates the subcategorisation frame.

- $possibleArgument(a) \wedge frame(a,+f) \wedge isArgument(a)$

C.3 *hasRole/2*

Bias: Captures the most common predicate-argument pairs.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge hasRole(p,a)$

Distance: Relates predicate and argument based on the distance.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge hasRole(p,a) \wedge ||p-a|| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge hasRole(p,a) \wedge ||p-a|| = [0,1,2,3,4,5,10]$

Predicate: Relates the information of the token of the predicate of a predicate-argument pair.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+cp) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+l) \wedge hasRole(p,a)$

Predicate and argument: Relates the information of predicate and argument tokens with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a,+pa) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge ppos(a,+pa) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge lemma(a,+la) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge hasRole(p,a) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a,+pa) \wedge hasRole(p,a) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a+1,+pa) \wedge hasRole(p,a) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a-1,+pa) \wedge hasRole(p,a) \wedge |p-a| = [0,1,2,3,4,5,10]$

Coarse POS tag: Relates the information of the Coarse POS tag around the predicate and argument tokens.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge cpos(p,+cp1) \wedge cpos(p-1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a+1,+cp4) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge cpos(p,+cp1) \wedge cpos(p+1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a-1,+cp4) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge cpos(p,+cp1) \wedge cpos(p-1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a-1,+cp4) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge cpos(p,+cp1) \wedge cpos(p+1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a+1,+cp4) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$

PP attachment: Relates the predicate with the PP when the preposition is *in*.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge lemma(m,+lm) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge hasRole(p,a)$

Frame features: Relates the subcategorisation frames between predicate and argument with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge frame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge unlabelFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge unlabelFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge unlabelFrame(p,a) \wedge hasRole(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge pathFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge pathFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge voice(p,+v) \wedge pathFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge pathFrame(p,a,+f) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge voice(p,+v) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(p,+lp) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge lemma(a,+la) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a,+pa) \wedge pathFrameDistance(p,a,d) \wedge hasRole(p,a) \wedge d = [0,1,2,3,4,5,10]$

Dependency features: Relates the dependency structure of a predicate and argument with the hasRole.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge dep(\_,a,+d) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge dep(\_,a,+d) \wedge voice(p,+v) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge dep(\_,a,+da) \wedge dep(\_,p,+dp) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge dep(\_,a,+da) \wedge dep(\_,p,+dp) \wedge voice(p,+v) \wedge hasRole(p,a) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge path(p,a,+p) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge path(p,a,+p) \wedge ppos(a,+pa) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge path(p,a,+p) \wedge slemma(a,+la) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge path(p,a,+p) \wedge cpos(a,+pa) \wedge lemma(p,+lp) \wedge hasRole(p,a)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge path(p,a,+p) \wedge slemma(a,+la) \wedge slemma(p,+lp) \wedge hasRole(p,a)$

WordNet features: It includes some WordNet feature (these formulae only appears in the original model).

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge wnet(p,+w) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge wnet(a,+w) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge wnet(p,+w) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge ppos(p,+pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge wnet(m,+w) \wedge role(p,a,+r)$

C.4 *role/3*

Predicate: Relates the information of the token of the predicate with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,+cp) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+l) \wedge role(p,a,+r)$

Predicate and argument: Relates the information of predicate and argument tokens

with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,+pp) \wedge ppos(a,+pa) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge role(p,a,+r) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge role(p,a,+r) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(a,+la) \wedge voice(p,+v) \wedge role(p,a,+r) \wedge |p-a| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(a,+pa) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$

Coarse POS tag: Relates the information of the Coarse POS tag around the predicate and argument tokens.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge cpos(p,+cp1) \wedge cpos(p-1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a+1,+cp4) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge cpos(p,+cp1) \wedge cpos(p+1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a-1,+cp4) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge cpos(p,+cp1) \wedge cpos(p-1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a-1,+cp4) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge cpos(p,+cp1) \wedge cpos(p+1,+cp2) \wedge cpos(a,+cp3) \wedge cpos(a+1,+cp4) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$

PP attachment: Relates the predicate with the PP when the preposition is *in*.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge lemma(m,+lm) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge role(p,a,+r)$

Frame features: Relates the subcategorisation frames between predicate and argument

with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge frame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge unlabelFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge unlabelFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge unlabelFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge pathFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge pathFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge voice(p,+v) \wedge pathFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge lemma(a,+la) \wedge pathFrame(p,a,+f) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge voice(p,+v) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(p,+lp) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge lemma(a,+la) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,+pp) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,+pp) \wedge ppos(a,+pa) \wedge pathFrameDistance(p,a,d) \wedge role(p,a,+r) \wedge d = [0,1,2,3,4,5,10]$

Dependency features: Relates the dependency structure of a predicate and argument with the role.

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge dep(\_,a,+d) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge dep(\_,a,+d) \wedge voice(p,+v) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge dep(\_,a,+da) \wedge dep(\_,p,+dp) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge dep(\_,a,+da) \wedge dep(\_,p,+dp) \wedge voice(p,+v) \wedge role(p,a,+r) \wedge |a-p| = [0,1,2,3,4,5,10]$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge path(p,a,+p) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge path(p,a,+p) \wedge ppos(a,+pa) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge path(p,a,+p) \wedge slemma(a,+la) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge path(p,a,+p) \wedge cpos(a,+pa) \wedge lemma(p,+lp) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge path(p,a,+p) \wedge slemma(a,+la) \wedge slemma(p,+lp) \wedge role(p,a,+r)$

WordNet features: It includes some WordNet feature (these formulae only appears in the original model).

- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge wnet(p,+w) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge wnet(a,+w) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge wnet(p,+w) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge ppos(m,+pm) \wedge role(p,a,+r)$
- $possiblePredicate(p) \wedge possibleArgument(a) \wedge palmer(p,a) \wedge ppos(p,+pp) \wedge ppos(a,IN) \wedge dep(a,m,\_) \wedge wnet(m,+w) \wedge role(p,a,+r)$

C.5 *sense/2*

Bias: Captures the most common sense-predicate pairs.

- $possiblePredicate(p) \wedge sense(p,+s)$
- $possiblePredicate(p) \wedge slemma(p,+lp) \wedge sense(p,+s)$

Bibliography

Steven Abney. Partial parsing via finite-state cascades. *Natural Language Engineering*, 4(2):337–344, 1996.

J.F. Allen, B. Miller, E. Ringger, and T. Sikorski. A robust system for natural spoken dialogue. In *Proceedings of 34th Annual Meeting of the Association for Computational Linguistics (ACL)*, 1996.

S. Bennacef, L. Devillers, S. Rosset, and L. Lamel. Dialog in the RAILTEL telephone-based system. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, pages 550–553, 1996.

C. Bennett and A.I. Rudnicky. The Carnegie Mellon Communicator Corpus. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, pages 341–344, 2002.

Rens Bod. Context-sensitive dialogue processing with the DOP model. *Natural Language Engineering*, 5(4):309–323, 2000.

R. Bonnema, G. van Noord, and G. Veldhuijzen van Zanten. Evaluation results NLP components OVIS, 1998. Technical Report.

Thorsten Brants. TnT - a statistical Part-of-Speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP)*, 2000.

Sabine Buchholz and Erwin Marsi. CoNLL-X Shared Task on multilingual dependency parsing. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2006.

Xavier Carreras and Lluís Márquez. Introduction to the CoNLL-2005 Shared Task: Semantic role labeling. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2005.

Wanxiang Che, Zhenghua Li, Yuxuan Hu, Yongqiang Li, Bing Qin, Ting Liu, and Sheng Li. A cascaded syntactic and semantic dependency parsing system. In *Proceedings of Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2008.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003.

Deborah A. Dahl and et al. ATIS 3 training data, 1994. Linguistic Data Consortium, Philadelphia.

Myroslava Dzikovska, Mary Swift, James Allen, and William de Beaumont. Generic parsing for multi-domain semantic interpretation. In *Proceedings of International Conference on Parsing Technologies (IWPT)*, 2005.

Giuseppe Di Fabbrizio, Dawn Dutton, Narendra Gupta, Barbara Hollister, Mazin Rahim, Giuseppe Riccardi, Robert Schapire, and Juergen Schroeter. AT&T help desk. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, 2002.

C. Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998.

M. Frampton and O. Lemon. Using dialogue acts to learn better repair strategies for spoken dialogue systems. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2008.

Kallirroi Georgila, James Henderson, and Oliver Lemon. Learning user simulations for information state update dialogue systems. In *Proceedings of Interspeech*, 2005a.

Kallirroi Georgila, Oliver Lemon, and James Henderson. Automatic annotation of communicator dialogue data for learning dialogue strategies and user simulations. In *Proceedings of DIALOR: The 9th workshop on the semantics and pragmatics of dialogue (SEMDIAL)*, 2005b.

J. Gimenez and Lluís Márquez. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of International Conference on Language Resources and Evaluation (LREC)*, 2004.

A. L. Gorin, G. Riccardi, and J. H. Wright. How may i help you? *Speech Commun.*, 23(1-2):113–127, 1997. ISSN 0167-6393.

N. Gupta, G. Tur, D. Hakkani-Tur, S. Bangalore, G. Riccardi, and M. Rahim. The AT&T spoken language understanding system. *IEEE Transactions on Speech and Audio Processing*, 4(1):213–222, 2006.

P. Haffner, G. Tur, and J. Wright. Optimizing SVMs for complex call classification. In *International Conference on Acoustic Speech and Signal Processing (ICASSP)*, 2003.

Yulan He and Steve Young. Semantic processing using the Hidden Vector State model. *Computer Speech and Languge*, 19:85–106, 2005.

Yulan He and Steve Young. Spoken language understanding using the Hidden Vector State model. *Speech Communication Special Issue on Spoken Language Understanding for Conversational Systems*, 48, No. 3-4:262–275, 2006.

Gang Ji and Jeff Bilmes. Dialog act tagging using graphical models. In *International Conference on Acoustic Speech and Signal Processing (ICASSP)*, 2005.

Richard Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for english. In *Proceedings of Nordic Conference of Computational Linguistics (NODALIDA)*, 2007.

Richard Johansson and Pierre Nugues. Dependency-based syntacticsemantic analysis with PropBank and NomBank. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2008.

B. J. Frey Kschischang and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. The WITAS Mult-Modal Dialogue System I. In *Proceedings of EuroSpeech*, 2001.

Oliver Lemon, Kallirroi Georgila, and James Henderson. Evaluating effectiveness and portability of reinforcement learned dialogue strategies with real users: the TALK TownInfo evaluation. In *Proceedings of IEEE/ACL Spoken Language Technology*, 2006a.

Oliver Lemon, Kallirroi Georgila, James Henderson, and Matthew Stuttle. An ISU dialogue system exhibiting reinforcement learning of dialogue policies: generic slot-filling in the TALK in-car system. In *Proceedings of Conference of the European Chapter of the Association of Computational Linguistics (EACL)*, 2006b.

F. Mairesse, M. Gasic, F. Jurcicek, S. Keizer, B. Thomson, K. Yu, and S. Young. Spoken language understanding from unaligned data using discriminative classification models. In *International Conference on Acoustic Speech and Signal Processing (ICASSP)*, To appear.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: the Penn Treebank. *Computational Linguistics*, 19, 1993.

Lluís Márquez, Xavier Carreras, Ken Litkowski, and Suzanne Stevenson. Semantic role labeling. *Computational Linguistics*, 34(2), 2008. Introduction to the Special Issue on Semantic Role Labeling.

A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. Annotating Noun Argument Structure for NomBank. In *Proceedings of International Conference on Language Resources and Evaluation (LREC)*, 2004.

Ivan Meza-Ruiz and Sebastian Riedel. Jointly identifying predicates, arguments and senses using Markov Logic. In *Proceedings of North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT) conference*, 2009. To appear.

Ivan Meza-Ruiz, Sebastian Riedel, and Oliver Lemon. Accurate statistical spoken language understanding from limited development resources. In *International Conference on Acoustic Speech and Signal Processing (ICASSP)*, 2008a.

Ivan Meza-Ruiz, Sebastian Riedel, and Oliver Lemon. Spoken language understanding in dialogue systems, using a 2-layer Markov Logic Network: improving semantic accuracy. In *Late Breaking Abstracts of Londial: The 9th workshop on the semantics and pragmatics of dialogue (SEMDIAL)*, 2008b.

Marvin Minsky. A framework for representing knowledge. Technical report, 1974. Source Technical Report: AIM-306.

Stephan Oepen, Dan Flickinger, Kristina Toutanova, and Christoper D. Manning. Lingo redwoods: A rich and dynamic treebank for HPSG. In *First Workshop on Treebanks and Linguistic Theories (TLT2002)*, 2002.

David S. Pallett, Jonathan G. Fiscus, William M. Fisher, John S. Garofolo, Bruce A. Lund, and Mark A. Przybocki. 1993 benchmark tests for the ARPA spoken language

program. In *Proceedings of the Human Language Technology Conference (HLT)*, pages 49–74, 1994.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31, 2005.

V. Punyakanok, D. Roth, and W. Yih. Generalized inference with multiple semantic role labeling systems. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2005.

Matthew Purver. Processing unknown words in a dialogue system. In *Proceedings of the 3rd SIGdial Workshop on Discourse and Dialogue*, pages 174–183, 2002.

Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77 (2), pages 257–286, 1989.

Lance A. Ramshaw and Mitchell P. Marcus. Text chunking using transformation-based learning. In *Proceedings of CoRR*, pages 82–94, 1995.

A. Ranta. Grammatical framework: A type-theoretical grammar formalism. *Journal of Functional Programming*, 2002.

Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1-3):151–175, 1999.

Matt Richardson and Pedro Domingos. Markov Logic Networks. *Machine Learning*, 62:107–136, 2006.

Sebastian Riedel. TheBeast: ML software, 2007. URL http://code.google.com/p/thebeast/.

Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *Proceedings Uncertainty in Artificial Intelligence conference (UAI)*, 2008.

Sebastian Riedel and Ivan Meza-Ruiz. Collective Semantic Role Labelling with Markov Logic. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2008.

Stephanie Seneff, Raymond Lau, and Joseph Polifroni. Organization, communication, and control in the Galaxy-II conversational system. In *Proceedings of Eurospeech*, pages 1271–1274, 1999.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Llus Mrquez, and Joakim Nivre. The CoNLL 2008 Shared Task on joint parsing of syntactic and semantic dependencies. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2008.

Charles Sutton and Andrew McCallum. An introduction to Conditional Random Fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. 2006.

Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. Joint learning improves semantic role labeling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL)*, 2005.

Gokhan Tur, Dilek Hakkani-Tr, and Ananlada Chotimongkol. Semi-supervised learning for spoken language understanding using semantic role labeling. In *Proceedings of ASRU-2005*, 2005.

Sebastian Varges and Matthew Purver. Robust language analysis and generation for spoken dialogue systems. In *Proceedings of the ECAI workshop on Development and Evaluation of Robust Spoken Dialogue Systems for Real Applications*, 2006.

Gert Veldhuijzen van Zanten. Semantics of update expressions, 1996. Technical Report 24, NWO Priority Programme Language and Speech Technology.

David Vickrey and Daphne Koller. Applying sentence simplification to the CoNLL-2008 Shared Task. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, 2008.

M. Walker and R. Passonneau. DATE: A dialogue act tagging scheme for evaluation of spoken dialogue systems. In *Proceedings of the Human Language Technology Conference (HLT)*, 2001.

Karl Weilhammer, Rebecca Jonson, Hakan Burden, Jost Schatzmann, Matt Stuttle, and Steve Young. Integratting multiple modalities into SLMs and parsing the output of SLMs, 2006. TALK public deliverables.

Yuk Wah Wong and Raymond J. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 960–967, 2007.

Wei-Lin Wu, Ru-Zhan Lu, Jian-Yong Duan, Hui Liu, Feng Gao, and Yu-Quan Chen. A weakly supervised learning approach for spoken language understanding. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, 2006.

Nianwen Xue and Martha Palmer. Calibrating features for semantic role labeling. In *Proceedings of Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2004.

Hui Ye and Steve Young. Clustering approach to semantic decoding. In *Proceedings of International Conference on Spoken Language Processing (ICSLP)*, 2006.

Luke Zettlemoyer and Michael Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of Joint Conference on Empirical Methods on Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

Deyu Zhou and Yulan He. A hybrid Generative/Discriminative Framework to train a semantic parser from an un-annotated corpus. In *Proceedings of International Conference on Computational Linguistics (COLING)*, 2008.